UiO : Department of Informatics
University of Oslo

# Improving dependency parsing using word clusters

Jostein Lien

Master's Thesis Spring 2015

# Improving dependency parsing using word clusters

Jostein Lien

9th February 2015

# Abstract

This thesis describes a semi-supervised approach to improving statistical dependency parsing using word clusters as features in MaltParser, a data-driven transition-based dependency parser. We experiment with using different clustering features for generating clusters, using the mini-batch variant of the K-means algorithm. The clusters are used as a source of additional information in an expanded feature model used by the MaltParser system. A baseline parser is applied to unlabeled text, generating a dependency representation of the text. K-means is then used for generating word clusters based on this dependency representation. We then re-train the parser with information on these clusters by using a cluster-informed feature model, having all the baseline feature functions, but expanded with addition cluster feature functions, making this an instance of semi-supervised learning. We report significantly improved parsing results when using a cluster-informed parser compared to the baseline parser, using both in-domain and out-of-domain data.

**Acknowledgements**

Foremost, I would like to thank my supervisors Lilja Øvrelid and Erik Velldal for their great patience and support through the work on this master thesis. Besides my supervisors, I would like to thank Stephan Oepen for providing additional technical support.

# Contents


**1 Introduction** **1**

**2 Dependency parsing** **3**
2.1 Constituent structures vs dependency structures . . . . . 3
2.1.1 Well-formedness and constraints on dependency graphs . . . . . 5
2.2 MaltParser . . . . . 6
2.2.1 Transition-based dependency parsing . . . . . 6
2.2.2 Parsing algorithms . . . . . 8
2.2.3 History-based feature models . . . . . 8
2.2.4 Feature models used in this thesis . . . . . 10
2.3 Summary . . . . . 13

**3 Clustering** **15**
3.1 Clustering . . . . . 16
3.2 Agglomerative hierarchical clustering . . . . . 16
3.2.1 Linkage criterions - cluster similarity . . . . . 17
3.2.2 Brown clustering algorithm . . . . . 18
3.3 K-means clustering . . . . . 20
3.3.1 Mini-batch K-means . . . . . 21
3.4 Hierarchical vs non-hierarchical clustering . . . . . 21
3.5 Evaluation of clusters . . . . . 22
3.6 Summary . . . . . 22

**4 Previous work** **25**
4.1 Supervised Dependency Parsing in English and Czech . . . 25
4.2 Word clusters for parsing French . . . . . 27
4.3 Syntactic word clustering for improving dependency parsing 28
4.4 Dependency parsing of web data . . . . . 30
4.5 The SANCL 2012 Shared task on web parsing . . . . . 32
4.5.1 NAIST dependency parsing for the SANCL shared task 34
4.6 Summary . . . . . 34

**5 Corpora and preprocessing** **37**
5.1 SANCL data sets . . . . . 37
5.2 Corpora for clustering . . . . . 38
5.2.1 Reuters Corpus Volume 1 . . . . . 38
5.2.2 Unlabeled SANCL domains . . . . . 39

5.3 Corpora for parsing . . . 39
5.3.1 OntoNotes Wall Street Journal . . . 39
5.3.2 Labeled SANCL domains . . . 40
5.3.3 Eng and Sel . . . 40
5.4 Gold standard part-of-speech tags vs automatically assigned part-of-speech tags . . . 40
5.5 Preprocessing the data . . . 41
5.5.1 Automatic part-of-speech tagging . . . 41
5.5.2 Lemmatization . . . 41

**6 Experiments** **43**
6.1 Experimental setup . . . 43
6.1.1 Model tuning and development . . . 43
6.1.2 Clustering . . . 45
6.1.3 Building a vocabulary . . . 47
6.1.4 Constructing the feature matrix . . . 50
6.1.5 Clustering process . . . 51
6.1.6 Parsing and evaluating . . . 52
6.2 Reuters clusters and the path-to-root-pos clustering feature . 54
6.2.1 Development . . . 54
6.2.2 Held out testing . . . 55
6.3 Reuters clusters and *dependency* clustering features . . . 56
6.3.1 Development . . . 56
6.3.2 Held out testing . . . 57
6.4 In-domain clusters and the path-to-root-pos clustering feature 59
6.4.1 Development . . . 59
6.4.2 Held out testing . . . 60
6.4.3 Comparing Reuters and in-domain held-out results . 62
6.5 In-domain clusters and *dependency* features . . . 63
6.5.1 Development . . . 63
6.5.2 Held out testing . . . 64
6.5.3 Comparing Reuters and in-domain results . . . 65
6.6 Comparison between the path-to-root-pos and *dependency* clustering features . . . 66
6.7 Clustering all the five unlabeled SANCL domains . . . 67
6.8 Testing with smaller vocabularies and agglomerative clustering . . . 69
6.9 Agglomerative clustering . . . 71
6.10 Comparing results with Øvrelid and Skjærholt . . . 73
6.11 Cluster examples . . . 75
6.12 Summary . . . 75

**7 Conclusion** **79**
7.1 Conclusion and further work . . . 79
7.1.1 Further work . . . 82

# List of Figures


2.1 Constituent parse tree example of sentence from Reuters . . 4
2.2 Dependency graph example of sentence from Reuters . . . . 4
2.3 CoNLL format example of sentence from Reuters . . . . . 5
2.4 ConLL file example annotated with cluster labels . . . . . 12

3.1 Hierarchical clustering example of lemmas from Reuters . . 18
3.2 Brown clustering example . . . . . 19

6.1 Dependency clustering features example . . . . . 46
6.2 Path-to-root-pos clustering features example . . . . . 47
6.3 Maltparser training example . . . . . 52
6.4 MaltParser parsing example example . . . . . 53
6.5 Clustering example of web data - part 1 . . . . . 76
6.6 Clustering example of web data - part 2 . . . . . 77

# List of Tables


2.1 MaltParser feature models used in this thesis . . . . . 11

5.1 Statistics for clustering data sets . . . . . 38
5.2 Statistics for SANCL data sets for parsing . . . . . 39
5.3 Statistics for Wall Street Journal split . . . . . 39
5.4 Parsing scores comparing gold vs auto Pos tags . . . . . 41

6.1 Most frequent lemmas from the Reuters corpus . . . . . 49
6.2 Most frequent lemmas from the All SANCL corpus . . . . . 50
6.3 Parsing scores for assessing impact from non-deterministic clustering 50,000 lemmas . . . . . 52
6.4 Development results on parsing WSJ 22 from clustering Reuters on path-to-root-pos features . . . . . 54
6.5 Development results on parsing Weblogs from clustering Reuters on path-to-root-pos features . . . . . 54
6.6 Development results on parsing Emails from clustering Reuters on path-to-root-pos features . . . . . 55
6.7 Held-out parsing results based on clustering Reuters on path-to-root-pos features . . . . . 55
6.8 Development results on parsing WSJ 22 from clustering Reuters on *dependency* features . . . . . 56
6.9 Development results on parsing Weblogs from clustering Reuters on *dependency* features . . . . . 57
6.10 Development results on parsing Emails from clustering Reuters on *dependency* features . . . . . 57
6.11 Held-out parsing results based on clustering Reuters on *dependency* features . . . . . 58
6.12 Comparing held-out results between *dependency* and path-to-root-pos geatures when clustering Reuters . . . . . 58
6.13 Number of clustering features based on unlabeled SANCL data for clustering . . . . . 59
6.14 Development results on parsing Weblogs from clustering in-domain on path-to-root-pos features . . . . . 60
6.15 Development results on parsing Emails from clustering in-domain on path-to-root-pos features . . . . . 60
6.16 Held-out parsing results with *Form_simple* model based on clustering in-domain on path-to-root-pos feature . . . . . 61

6.17 Held-out parsing results with *Form_all* model based on clustering in-domain on path-to-root-pos features form-all . 61
6.18 Comparing held-out parsing with *Form_simple* and *Form_all* models based on clustering in-domain on path-root-pos features . . . . . 62
6.19 Comparing held-out parsing results based on clustering Reuters and in-domain on path-root-pos features . . . . . 62
6.20 Clustering features statistics for unlabeled SANCL data using *dependency* features . . . . . 63
6.21 Development results on parsing Weblogs based on clustering in-domain on *dependency* features . . . . . 64
6.22 Development results on parsing Emails based on clustering in-domain on *dependency* features . . . . . 64
6.23 Held-out parsing results with the *Form_simple* model based on clustering in-domain on *dependency* features form-simple 64
6.24 Held-out parsing results with the *Form_all* model based on clustering in-domain on *dependency* features . . . . . 65
6.25 Comparing held-out parsing results based on clustering in-domain on *dependency* features . . . . . 65
6.26 Comparing held-out parsing results based on clustering Reuters and in-domain on *dependency* features . . . . . 65
6.27 Comparing all Reuters and in-domain held-out results . . . 66
6.28 Comparing held-out parsing results based on clustering all SANCL domains concatenated . . . . . 67
6.29 Comparing All SANCL and in-domain held-out-results . . . 68
6.30 Parsing scores for assessing impact from non-deterministic clustering comparing 10,000 and 50,000 lemmas . . . . . 69
6.31 Comparing held-out parsing results based on clustering 10,000 and 50,000 lemmas from Reuters on *dependency* features 70
6.32 Comparing held-out parsing results based on clustering 10,000 and 50,000 lemmas from in-domain based on *dependency* features. . . . . 71
6.33 Comparing held-out parsing results based on clustering 10,000 and 50,000 lemmas from all SANCL domains concatenated . . . . . 71
6.34 Comparing held-out parsing results based on K-means and agglomerative clustering on 5,000 lemmas from Reuters . . . 72
6.35 Comparing our held-out parsing results with parsing based on Brown clusters (Øvrelid, Skjærholt) with gold PoS tags . 73
6.36 Comparing our held-out parsing results with parsing based on Brown clusters (Øvrelid, Skjærholt) with auto PoS tags . 74

# Chapter 1

# Introduction

Parsing web text has posed several challenges for parsers trained on edited newswire text, due to a great variation both in terms of different genres and level of formality. Web language may range from edited articles to more informal domains, such as user fora, weblogs, and social media. Lexical statistics used by a parser will become less reliable when parsing different domains, as the amount of unknown words increases (Øvrelid and Skjærholt, 2012). Since the training data typically is the Wall Street Journal of the Penn Treebank (Marcus et al., 1993), parsing web language has proven to be difficult, due to a mismatch between the training data and the data to be parsed. Some syntactic constructions are more frequent in web texts, such as questions, imperatives and sentence fragments. Spelling mistakes, use of slang, inconsistent use of punctuations and ungrammatical sentences is also often common in web language (Petrov and McDonald, 2012).

In recent years, there has been an increasing interest in a particular instance of parsing known as *dependency parsing* relying upon a *dependency-based representation*. The CoNLL (Conference on Computational Natural Language Learning) 2006 (Buchholz and Marsi, 2006) and CoNLL 2007 (Mcdonald et al., 2007) shared tasks were both devoted to multilingual dependency parsing. Dependency-based approaches to syntactic parsing are applied to a wide range of languages and new methods are continually emerging.

MaltParser is a language-independent parsing system for *data-driven dependency parsing*, having the advantage of being both robust and efficient (Nivre et al., 2007). It was one of the systems performing best on multilingual dependency parsing in the CoNLL 2006 and CoNLL 2007 shared tasks (Nivre and Hall, 2008). An advantage of data-driven methods in natural language processing, is that development time is often shorter than other resources, such as lexicons and grammars that are hand-crafted (Nivre et al., 2006). However, a data-driven approach will need a *treebank* (a collection of sentences annotated with the correct parse) which is expensive to produce. MaltParser is a tool taking advantage of the data-driven aspect, while having a modest demand on syntactically annotated data (Nivre et al., 2006).

A core task in dependency parsing is the assignment of word-to-word relations. Vital information is provided by statistics about these word relations in the training data, but a problem is that these statistics are often sparse. There has been an increasing amount of work for finding generalizations over the distribution of words and using different kinds of lexical categories in parsing (Øvrelid and Skjærholt, 2012). Relevant previous work in this direction is based on the use of word clusters in parsing. Using information about word clusters estimated from unlabeled data having the same domain as the data to be parsed, may help to reduce the loss in parsing performance that is expected from using a parser trained on a treebank from a different domain. As argued by Koo et al. (2008), the lexical statistics important to disambiguation in parsing is often sparse, and modeling relationships on a more general level than the words themselves may then be helpful (Koo et al., 2008). Candito and Seddah (2010) argued that word clusters may be a useful way of handling the problem of lexical data sparseness, since counts on clusters are more reliable and gives better probability estimates. Using word clusters may also help to handle the problem of mismatch between vocabularies in an out-of-domain corpus and the original treebank. Given the external corpus used to generate word clusters, the clusters work as an intermediary between the words in the treebank and the words in the out-of-domain corpus (Candito and Seddah, 2010).

The aim of this thesis is to study the effect of using cluster-based features in a MaltParser feature model, to see how these features contribute to parsing performance compared to a baseline feature model. We perform parsing experiments on several different data sets, including the Wall Street Journal and texts from various web domains. In all experiments, the parser is trained on the Wall Street Journal. By performing both clustering and parser testing on different domains, including news stories and more informal domains of web language, such as weblogs and newsgroups, we see how these changes in domains affect parsing performance.

The thesis is structured as follows. Chapter 2 provides an overview of dependency parsing and MaltParser, the parsing system we use in our experiments, as well as the various feature models used by the system. Chapter 3 provides an overview of clustering, an approach within machine learning that forms the basis of our experiments. In chapter 4, we give an overview of related work on using word clusters for improving statistical parsers. After these introductory chapters, we will focus on the main parts of this thesis. Chapter 5 details the data sets we use in our experiments for generating clusters and performing parsing, as well as the preprocessing of these data sets. Chapter 6 details the experiments we perform, reporting the parsing results under a variety of experimental conditions. In chapter 7, we provide a summary of the work performed and some thoughts regarding future directions.

# Chapter 2

# Dependency parsing

This chapter will give an overview of some central topics that is the basis for the parsing experiments described in chapter 6. Starting with the notion of dependency parsing, we explain how it differs from constituent parsing. This is followed by an introduction to MaltParser, a system for data-driven dependency parsing, which we use in our experiments. More specifically, we describe its use of parsing algorithms and feature models.

## 2.1 Constituent structures vs dependency structures

The notion of *constituency* is that a group of words can be seen as a phrase or unit, called a constituent. Each constituent is labeled with a syntactic category, such as NP for a noun phrase. The words included in a constituent acts as a single syntactic unit. Noun phrases may consist of a single word, such as *Horse*, or more than one words, such as *The horse*. The latter NP constituent can be divided into two other constituents, such as a determiner (Det) and a noun (N). Given a string of words representing a sentence, this can then be modelled as a tree diagram, known as a *constituent structure tree*. A sentence is then shown as both a linear sequence of words, and a hierarchical structure with constituents nested within other constituents, revealing the hierarchical structure of syntactic categories (Fromkin et al., 2011). The tree diagram in figure 2.1 shows an example of a graphic representation of sentence structure, using the example sentence: *Recovery excitement brings Mexican markets to life.* This sentence is the headline of the first news story from the Reuters corpus.

An alternative approach of modelling syntactic sentence structure, is the notion of *dependency*. The idea is to investigate how words are related to other words in a sentence. A dependency relation between two words is considered a *binary asymmetric relation* between a *head word* and a *dependent word*. A head token may have more than one dependent token. The syntactic structure of a sentence is then a set of relations between the words in the sentence. These relations between the words, are known as *dependencies*. The main difference between constituency representations and dependency representations, is then the lack of constituent nodes in the dependency construction (Nivre, 2005).

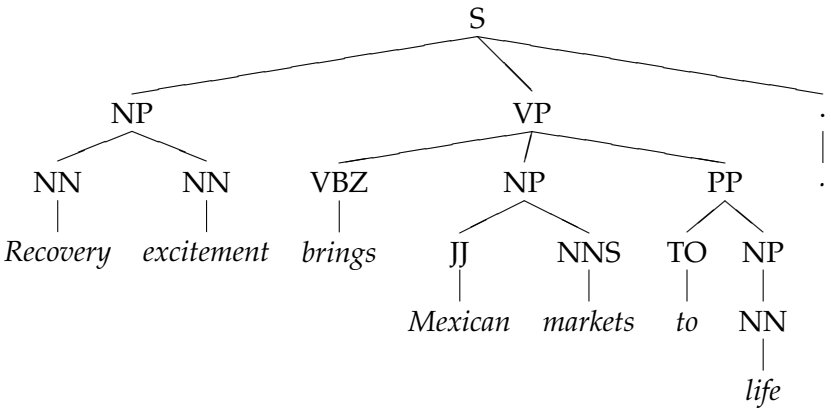


Figure 2.1: *A constituent parse-tree of the sentence 'Recovery excitement brings Mexican markets to life.' Starting from the top of the tree, the finer-grained constituent units of the sentence are gradually emerging.*

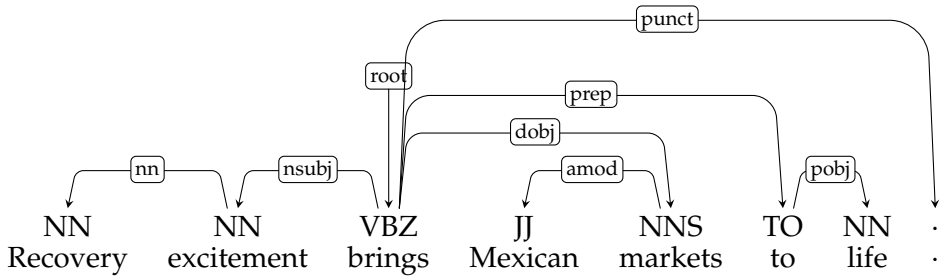


Figure 2.2: *A dependency graph of the sentence 'Recovery excitement brings Mexican markets to life.' The syntactic structure of the sentence is described as a set of binary relations between the words, more specifically a set of dependencies between the words.*

Given the sentence used as an example in the constituent approach, we can represent this with a dependency structure, known as a *dependency graph*.

Figure 2.2 shows a *labeled directed graph*, consisting of:

- A set of nodes $V$, representing the linear positions of the words in a sentence.
- A set of labeled arcs $E$ between the words. An arc can be considered a triple *(w1, l, w2)*, where w1 represents the head and w2 represents the dependent in a dependency construction, and $l$ is the dependency label between the two, e.g. *(excitement, nn, Recovery)*

The sentence shown in the constituent and dependency representations, is also shown in the CoNLL data format in figure 2.3. The word *Recovery* has token number two as its head, which is the word *excitement*. Likewise,

```
1 Recovery recovery     NN NN _    2 nn _ _
2 excitement excitement NN NN _    3 nsubj _ _
3 brings bring          VBZ VBZ _  0 root _ _
4 Mexican Mexican       JJ JJ _    5 amod _ _
5 markets market        NNS NNS _  3 dobj _ _
6 to to                 TO TO _    3 prep _ _
7 life life             NN NN _    6 pobj _ _
8 . .                   . . _      3 punct _ _
```

Figure 2.3: *The CoNLL representation of the sentence 'Recovery excitement brings Mexican markets to life.' .*

*excitement* has the third token which is the root word *brings* as its head, connected by the dependency relation *nsubj*, creating the triple *(brings, nsubj, excitement)*.

For defining a dependency relation between a head and a dependent in a dependency construction, some criteria are proposed (Nivre, 2005), (Zwicky, 1985):

- The head is mandatory while the dependent may be optional.
- The syntactic category in the construction is determined by the head, and may replace the construction.
- The head selects the dependent, and determines whether the dependent is optional or mandatory.

Furthermore, there is a distinction between dependency in *endocentric* and *exocentric* constructions (Bloomfield, 1933):

In endocentric constructions, the head may replace the whole dependency construction, without disrupting syntactic structure. An example of this is a construction consisting of a noun which is a head, and an adjective which is the dependent, such as *(markets, amod, Mexican)* from the example in figure 2.2. The dependent may be removed, without disrupting the syntactic structure of the whole sentence.

In exocentric constructions, on the other hand, the head can not replace the whole construction. This is seen in the construction *(to, pobj, life)* from figure 2.2.

### 2.1.1 Well-formedness and constraints on dependency graphs

A dependency graph is *well-formed* when having the following constraints (Nivre et al., 2007):

- There is a special node 0 called the *root node* not corresponding to any token in the sentence. It is normally the only root of the sentence.
- The dependency graph is *weakly connected*, meaning that there is a path from each node to each other node in the graph, without

accounting for direction on the arcs between nodes. The root node is crucial for connectedness, achiving robust parsing, since it ensures that there will always be a single entry point in the graph.

Some other common constraints on dependency graphs (Nivre et al., 2007) are:

- *Single-head constraint*, meaning that each token has at most one head.
- *Acyclic*, if there exist an arc $(i \rightarrow j)$, there is no arc $(j \rightarrow i)$.
- A dependency graph is *projective* if, when the words are written in linear order, the dependency arcs between the words can be written above the words without crossing (Bird et al., 2009). More formally, for every arc $(i \rightarrow j)$, then $(i \rightarrow k)$, for every node k such that i < k < j or j < k < i.

## 2.2 MaltParser

MaltParser is a *data-driven parser for inductive dependency parsing* (Nivre et al., 2007). A traditional parser for grammar-driven parsing constructs a parser using a grammar G, defining the language L(G) that can be parsed. In data-driven parsing, no grammar is used. Instead, labeled dependency treebank data (manually annotated sentences with the correct dependency graph annotations) in the dependency format for a given language is used for inducing a parser for that language. The MaltParser system can be run in two modes (Nivre et al., 2007):

In *learning mode*, the input is a dependency treebank. Given the user-specification of a *learning algorithm, parsing algorithm and a feature model*, a classifier for prediction of parser actions is induced. The correct parser transition sequences (transitions between states) are reconstructed for creating training data, and the classifier is trained on this data.

In *parsing mode*, the input consist of a set of test sentences, a trained classifier induced during learning, and the parsing algorithm and feature model specified in learning mode. Using the classifier as a guide, the test sentences are then parsed, constructing a dependency graph for each sentence. The classifier guides the parser at non-deterministic choice points, using a history-based feature model, described in section 2.2.3.

### 2.2.1 Transition-based dependency parsing

Two main approaches in data-driven dependency parsing are *graph-based* and *transition- based* models. The basic idea in graph-based dependency parsing is to learn a model through global training for assigning a score to an entire dependency graph for an input sentence (McDonald and Nivre, 2011). Given the induced model, parsing is then performed by exhaustive searching for the dependency graph with the highest score, through the search space of candidate dependency graphs for the sentence.

In the transition-based approach to dependency parsing employed by MaltParser, a model is instead learned through local training for scoring transitions from one parser state to a new state. The model is induced for predicting the next state, given the parse history. Parsing is then performed by using the highest scoring transition between the states, until a complete dependency graph is derived. In other words, the induced model guides the parser where more than one transition is possible, choosing the highest scoring transition by greedy searching, constructing an optimal dependency graph, through an optimal transition sequence.

In MaltParser, parsing can be described as a set of transitions between *parse configurations*. Algorithms most commonly used in dependency parsing, are seen as variants of the *shift -reduce algorithm*. The input sentence is analyzed from left to right, while two data structures, a *stack* for holding partially processed tokens in the construction of a dependency graph, and a *list* of the remaining tokens from the input sentence (Nivre et al., 2007).

A parse configuration is defined as a triple <S, I, G>:

- *S* is the stack of partially processed tokens, being either candidates as heads or dependents for dependency arcs.
- *I* is the list of remaining tokens from the input sentence, starting from left to right.
- *G* is the dependency graph currently under construction.

During parsing, the parser predicts the next transition (parse action) using the trained model as a guide, given the current parse configuration. For example, in Nivres's algorithm using the so-called arc-eager mode, four parser actions are employed (Nivre and Hall, 2008):

1. *SHIFT*, for pushing the next token from the input sentence onto the stack, as long as there are remaining tokens in the input. This parse action is needed for handling tokens having their heads to the right, or for tokens that should be attached to the root node.
2. *RIGHT-ARC*, for making a right-arc with dependency label *l* from token *w1* currently on top of the stack, to the next token *w2* in the input sentence, making *w1* a head and *w2* a dependent. The dependency relation *(w1, l, w2)* is created, and *w2* is pushed onto the stack.
3. *LEFT-ARC*, for making a left-arc with dependency label *l* from the next input token *w2* in the input sentence, to the token *w1* on top of the stack, making *w2* a head, and *w1* a dependent. The dependency relation *(w2, l, w1)* is created, and *w1* is popped from the stack.
4. *REDUCE*, for removing the token currently on top of the stack. This can only be done if that token is assigned to a head token. This parsing action is needed for removing a token that was pushed onto the stack with the Right-arc transition, and has since found all the right dependents of the token.

### 2.2.2 Parsing algorithms

The parsing algorithms available in MaltParser are variants of shift-reduce algorithms, and made deterministic by the classifier which makes decisions at non-deterministic choice points. In MaltParser version 1.7.2, there are three families of parsing algorithms [1]:

- *Nivre's algorithm* is limited to projective dependency graphs and has a linear time complexity. There are two modes available: *arc-eager* and *arc-standard*. It uses two data structures: A *stack* for holding partially processed tokens in the construction of the dependency graph, and a *list* working as a buffer for holding the remaining tokens in the input sentence.

- *Stack algorithms*, operates with a stack and a list data structure, as in Nivre's algorithm. The difference is that the Stack algorithms add dependency arcs between the two tokens currently on top of the stack, while Nivre's algorithm adds dependency arcs between the token on top of the stack and the next token from the input. The Stack algorithms has three data structures: A *stack* of the partially processed tokens during the construction of the dependency graph, a *list* holding all the tokens that have been on the stack, and a *lookahead list* for holding all the tokens that have not been on the stack. When run in projective mode (stackproj), the stack algorithm is limited to projective dependency graphs and uses the same set of transitions as in Nivre's arc-standard mode. In *Eager* (stackeager) and *Lazy* (stacklazy) variants of the Stack algorithm, it is possible to derive non-projective dependency graphs by applying swap transitions.

- *Covington's algorithm* has no restrictions on the dependency structure, and has a quadratic time complexity. The algorithm works by trying to link each new token to each preceding token. When run in *projective* mode, it is limited to projective dependency graphs, while in *non-projective* mode, non-projective dependency graphs are allowed.

In the parsing experiments in this thesis, we use MaltParser with the stacklazy algorithm in the same the vein as Øvrelid and Skjærholt (2012).

### 2.2.3 History-based feature models

The idea of a *history-based feature model* comes from a specific parse history being represented as a sequence of *attributes*, corresponding to a *feature vector*. A *feature model* is then defined as a sequence of *feature functions*, where each function returns an attribute as a feature, considered relevant from the parse history. MaltParser uses the feature models for predicting the next parse action in the derivation of the dependency graph by combining features from the partially constructed dependency graph with features from the input sentence. By applying the feature functions given

[1] `http://www.maltparser.org/userguide.html#parsingalg`

the current *parser configuration*, a feature vector (sequence of features) is then used by a trained classifier for predicting the next parse action. Features considered important in dependency parsing are attributes of the tokens in the input sentence, and these attributes are divided into the following two categories (Nivre et al., 2007):

*Static attributes* remain constant during the parsing of the sentence, most importantly the word form of the token, but also attributes being the result of various preprocessing steps, such as part-of-speech tagging and lemmatization.

*Dynamic attributes*, on the other hand, are given by the partially constructed dependency graph, such as the dependency label relating a token to its head.

In the specification of history-based feature models, there is a need for referring to the attributes for the tokens in the current parse configuration. A set of *address functions*, are used for retrieving specific tokens from the current parse configuration (Nivre et al., 2007):

- $\sigma(i)$, retrieves token number *i* starting from the top of the stack.
- $\tau(i)$, retrieves token number *i* from the remaining input sentence.
- *h(i)*, retrieves the head for token number *i* in the dependency graph.
- *l(i)*, retrieves the leftmost child for token number *i* in the dependency graph.
- *r(i)*, retrieves the rightmost child for token number *i* in the dependency graph.

These address functions can then be combined into more complex functions, such as *r(h)*($\sigma$0), which retrieves the rightmost dependent of the head of the token currently on the top of the stack. The feature functions are then defined by using the address functions as input to the attribute functions: *d(h)*($\sigma$0), retrieves the dependency label of the head of the token currently on the top of the stack. Feature functions not defined for a specific parser configuration, are assigned a *nil* value (Nivre et al., 2007).

A feature model is defined in an XML file, where each feature function is on a separate line. For each parsing algorithm, MaltParser has a default feature model specification. The feature functions are defined for the CoNLL data format. Some examples of these functions from table 2.1 are:

- The feature template $S_0p$ corresponds to the part-of-speech tag for the token currently on the top of the stack.
- The feature template $S_{0l}d$ corresponds to the dependency label for the leftmost dependent for the token currently on top of the stack.
- The feature template $S_0wL_0w$ corresponds to a merge function, where a feature value is created by merging the word form for the token currently on the top of the stack, with the word form for the next token in the lookahead list.

### 2.2.4 Feature models used in this thesis

Table 2.1 shows the various feature models used by (Øvrelid and Skjærholt, 2012). A baseline feature model as described in (Foster et al., 2011), for parsing web 2.0 data, is using the stacklazy parsing algorithm. In addition to the baseline feature model, there are three *extended feature models.* They include the same feature templates as the baseline model, but are also extended with additional features for extracting, for instance, $S_0l$ for the cluster label for the token currently on the top of the stack.

We see from table 2.1 that the *PoS_simple* model includes all the features from the baseline model, as well as introducing some cluster-based features making use of a word's cluster label id from column 11 in the CoNLL data format. Feature functions extracting cluster labels for tokens in both the stack and the lookahead list is included, for instance the $S_0l$ and $L_0l$ features.

*Form_simple* has all the baseline features, in addition to some cluster-based features which are all included in the *PoS_simple* model as well. *Form_simple* does, however, lack the features $S_3l$, $L_2l$, $I_0l$ and $S_{0r}l$, which are included in the *PoS_simple* model.

*Form_all* includes the same features as the baseline and *Form_simple* models. In addition, this is also the only model of the three extended models that uses conjunctive features, such as $S_0lL_0l$ which merges the cluster label from the token on top of the stack with the cluster label for the next token in the Lookahead list.

The three data structures used in the stacklazy algorithm, has the following abbreviations:

- $S$, the stack holding the partially processed tokens.
- $L$, the lookahead list holding the tokens that have not yet been on the stack.
- $I$, the list of tokens that have been on the stack.

The various token attributes are abbreviated by:

- $p$, part-of-speech tag.
- $w$, word form.
- $d$, dependency label
- $l$, lexical category, in our case the cluster label.

The feature $L_0w$ will then be the word form of the first token in the lookahead list, while the feature $S_{1r}l$ is the cluster label (or lemma) of rightmost dependent of the second token on the stack. In the *Form_all* model, the feature $L_0pL_0l$ correspond to the merging of the part-of-speech and cluster label of the first token in the lookahead list, into one single feature.

A feature model employed by MaltParser has feature functions for extracting information from a sentence in the CoNLL data format, such

| **Feature set** | **Feature templates** |
|---|---|
| *Baseline* | $S_0p$, $S_1p$, $S_2p$, $S_3p$, $L_0p$, $L_1p$, $L_2p$, $I_0p$, $S_{0l}p$, $S_{0r}p$, $S_{1r}p$, $S_{0l}d$, $S_{1r}d$, $S_0w$, $S_1w$, $S_2w$, $L_0w$, $L_1w$, $S_{0l}w$, $S_{1r}w$, $S_0pS_1p$, $S_0wL_0w$, $S_0pS_0w$, $S_1pS_1w$, $L_0pL_0w$, $S_{1r}dS_{0l}d$, $S_{1r}pS_{1l}p$, $S_0pS_1pL_0p$, $S_0pS_1pS_2p$, $S_0pL_0pL_1p$, $L_0pL_1pL_2p$, $L_1pL_2pL_3p$, $S_0pL_0pI_0p$, $S_1pS_{1l}dS_{1r}d$ |
| *PoS_simple* | + $S_0l$, $S_1l$, $S_2l$, $S_3l$, $L_0l$, $L_1l$, $L_2l$, $I_0l$, $S_{0l}l$, $S_{0r}l$, $S_{1r}l$ |
| *Form_simple* | + $S_0l$, $S_1l$, $S_2l$, $L_0l$, $L_1l$, $S_{0l}l$, $S_{1r}l$ |
| *Form_all* | + $S_0l$, $S_1l$, $S_2l$, $L_0l$, $L_1l$, $S_{0l}l$, $S_{1r}l$, $S_0lL_0l$, $S_0pS_0l$, $S_1pS_1l$, $L_0pL_0l$, |

Table 2.1: *Feature models for the baseline and the re-trained parser, where p = PoS-tag, w = word form, d = dependency label in the graph constructed so far (if any), and l = cluster label. MaltParser's stacklazy algorithm operates over three data structures: a stack (S) of partially processed tokens, a list (I) of nodes that have been on the stack, and a "lookahead" list (L) of nodes that have not been on the stack. We refer to the top of the stack using $S_0$ and subsequent nodes using $S_1, S_2$, etc., and the leftmost/rightmost dependent of $S_0$ with $S_{0l}$ / $S_{0r}$.*

as the part-of-speech tag or word form for a token in the sentence. An example of a sentence from the Wall Street Journal corpus (section 23) in this format is shown in figure 2.4. Sentences are separated by a blank line and each token in a sentence consists of 11 fields. From left to right, these are:

1. Token id, starting at 1 for each new sentence.
2. Word form.
3. Lemmatized word form
4. Coarse-grained part-of-speech tag.
5. Fine-grained part-of-speech tag.
6. Syntactic or morphological features.
7. Token id for the head for this token.
8. Dependency relation between this token and its head.
9. Projective head for this token.
10. Dependency relation between this token and its projective head.
11. Cluster label id for this token.

```
 1 Oil oil                NN NN _ 2 nn _ _ 20
 2 company company        NN NN _ 3 nn     _ _ 84
 3 refineries refinery NNS NNS _ 4 nsubj _ _ _
 4 ran run                VBD VBD _ 0 root _ _ 89
 5 flat flat              JJ JJ _ 6 advmod _ _ 94
 6 out out                RP RP _ 4 advmod _ _ _
 7 to to                  TO TO _ 8 aux _ _ _
 8 prepare prepare        VB VB _ 4 xcomp _ _ 42
 9 for for                IN IN _ 8 prep _ _ _
10 a a                    DT DT _ 14 det _ _ _
11 robust robust          JJ JJ _ 14 amod _ _ 94
12 holiday holiday        NN NN _ 13 nn _ _ 52
13 driving driving        NN NN _ 14 nn _ _ 10
14 season season          NN NN _ 9 pobj   _ _ 82
15 in in                  IN IN _ 14 prep _ _ _
16 July July              NNP NNP _ 15 pobj _ _ 9
17 and and                CC CC _ 16 cc _ _ _
18 August August          NNP NNP _ 16 conj _ _ 9
19 that that              WDT WDT _ 22 nsubj _ _ _
20 did do                 VBD VBD _ 22 aux _ _ _
21 n't n't                RB RB _ 22 neg _ _ _
22 materialize materialize VB VB _ 14 rcmod _ _ _
23 . .                      . . _   4    punct _ _ _
```

Figure 2.4: *Example of a sentence from the Wall Street Journal section 23 in the CoNLL data format, where each token consists of 11 fields. The value to the far right is the cluster label id associated with that word.*

The cluster-based feature models we experiment with has the same feature functions as the baseline model, but will in addition also define

feature functions for extracting the cluster label for a token. We will then investigate what the effect these extra cluster-based features have on parsing performance, compared to only using the baseline features. For the purpose of this thesis, the $11^{th}$ column in the CoNLL format has been added for recording a word's cluster label id.

In order to use the cluster-based features, we need first to define a vocabulary of words. We then perform clustering of these words, that is, grouping words together so that words within a cluster are as similar as possible to each other, and as dissimilar as possible to words in other clusters. Each word within a specific cluster will then be associated with the same cluster label. Now that each word is associated with a cluster label, this label is written into the $11^{th}$ column in the CoNLL data format for the files that we parse in MaltParser. We choose to cluster only certain words, such as verbs, adjectives and nouns, so it is not the case that all the words in a sentence will be associated with a cluster label. From figure 2.4, we see that the adjectives *flat* and *robust* belongs to cluster 94, while the nouns *July* and *August* belongs to cluster number nine. The vocabulary we cluster is based on a data set containing a number of sentences within a given domain, such as news stories or various web domains. Some things to take into consideration here is the number of words we choose to cluster (vocabulary size), what domain the words belongs to, size of the data set used for retrieving the vocabulary, and the frequency of each word, as this is may affect parsing performance. Other considerations that may affect parsing performance, are what clustering algorithm we use and the choice of parameter values input to the algorithm, as well as *cluster granularity*, the number of clusters defined over the vocabulary. Other important parameters are the features used for generating clusters and the features used by MaltParser.

## 2.3 Summary

We have in this chapter given a description of some of the subjects regarding dependency parsing and the use of the MaltParser system that we use in our experiments, focusing on the parsing algorithms and features models employed by the system. During the model tuning and development part of the experimental process, we try out the different feature models; *PoS_simple*, *Form_simple* and *Form_all* , and choose the one who perform best and then compare its performance to the baseline model during held-out testing. The three aforementioned feature models makes use of *cluster-based features*, more specifically using a word's cluster label as an additional feature in the parsing process. The next chapter gives an overview of the machine learning technique known as *clustering*. This plays a fundamental part in our experiments.

# Chapter 3

# Clustering

This chapter will explain a few machine learning techniques, with an emphasis on *clustering*, which is a vital part of the experiments reported in chapter 6. We will first give an overview of some basics within machine learning, then move on to a more detailed explanation of some instances of clustering, more specifically *hierarchical agglomerative clustering* and *K-means* clustering.

A common task in machine learning is *classification*. This is an instance of *supervised learning*, where classes are predefined and objects are then assigned to them. A training set consisting of pairs of an object and its correct class is provided as input to a learning algorithm that returns a learned classification function. This function is then given a set of objects (test data) where the correct class for the objects are unknown, and assigns them correctly or incorrectly to one of the defined classes, replicating the supervised learning imposed on the data (Manning et al., 2008).

*Unsupervised learning*, on the other hand, where *clustering* is the most common method, means that there is no human expert having assigned objects to classes. Clustering algorithms group a set of objects into clusters (subsets). The goal is to make clusters where the objects within a cluster are as similar as possible, and objects in a cluster being as different as possible from objects in other clusters. The distance measure is provided as input to a clustering algorithm. This measure will affect the outcome of clustering, as different distance measures provides different clusterings. A distinction is made between *hard clustering* algorithms where each object to be clustered is a member of exactly one cluster, and *soft* clustering algorithms, where an object may be member of many clusters with various degrees of membership (Manning et al., 2008).

In *semi-supervised learning*, both labeled and unlabeled data are provided for learning. This method falls between supervised and unsupervised learning. A motivation for using semi-supervised learning is when a limited amount of labeled training data and a large amount of unlabeled data is available (Manning et al., 2008). The parsing model we use in our experiments in chapter 6, is an example of semi-supervised learning. We apply a baseline dependency parser to unlabeled data, converting these data to a dependency representation. The clustering algorithm we use will

then generate word clusters based on cluster-features of words from this dependency representation. A parser is then re-trained on labeled data with cluster ids for the words in these clusters, using a cluster-based feature model with feature functions recording information about the word's cluster ids.

## 3.1 Clustering

A distinction is made between *flat clustering* and *hierarchical clustering* algorithms. Flat clustering produces a specified number of clusters, with no structure relating different clusters to each other. Algorithms that produces flat clusters are usually iterative, starting with an initial set of clusters which are improved by reassigning objects to clusters through several iteration steps .

While *flat clustering* creates a flat unstructured set of unrelated clusters, requiring a specific number of clusters as input, *hierarchical clustering* creates a hierarchy of clusters, having the form of a tree structure. This is in some sense more informative than flat clustering, and there is no requirement of specifying the number of clusters. As further discussed below, however, if the set of nested partitions is to be converted to a single partition of flat clusters, the number of clusters still need to be specified. Algorithms for hierarchical clustering are divided into *agglomerative hierarchical clustering* (bottom-up clustering) and *divisive clustering* (top-down clustering). In divisive clustering, the objects to be clustered are contained in one set, and by recursively applying a flat clustering algorithm (like K-means ), clusters are split until all the objects are contained in its own cluster (Manning et al., 2008). We will in the following investigate bottom-up clustering in more detail.

## 3.2 Agglomerative hierarchical clustering

By using a bottom-up clustering approach, each object is initially contained in its own cluster. Pairs of clusters are then merged (agglomerated), and this is performed until all the clusters are merged into one cluster containing all the objects. A *dendrogram* is a tree diagram for visualizing the merging of clusters from the bottom to the top. This is a tree-like structure, where the leaves of the tree are the objects to be clustered. Each node in the tree corresponds to a cluster containing all the leaves that can be reached from that node, by traversing a path downwards. An example of such a dendrogram is shown in figure 3.1. On the y-axis, the level of cluster similarity is shown. When cutting the tree at, for instance, y = 0.6, only clusters with a minimum combination similarity of 0.6 are kept. Although there is no requirement to specify the number of clusters in hierarchical clustering, there are in some situations useful to have a partition of disjoint clusters as in flat clustering (Manning et al., 2008). Then the cluster hierarchy is cut at some point. Some criteria for this cutting point are (Manning et al., 2008).:

- Specify the number of clusters and select the cutting point that gives that number of clusters.

- Specify a level of similarity for cutting. As this level increases, the higher is the number of clusters.

- Cut the dendrogram where there is a large gap between two successive combination similarities

### 3.2.1 Linkage criterions - cluster similarity

There are a few different algorithms for agglomerative clustering corresponding to different ways of measuring the similarity between clusters: (Manning et al., 2008)

- *Single-link clustering*, where the similarity of two clusters is the similarity of their most similar objects.

- *Complete-link clustering*, where the similarity of two clusters is the similarity of their most dissimilar objects.

- *Group-average agglomerative clustering* computes the average similarity of all pairs of objects, also pairs from the same cluster, while excluding self-similarities.

- *Centroid clustering* computes similarity of two clusters by computing the similarity of cluster centroids. The difference between centroid clustering and group-average agglomerative clustering, is that the centroid method excludes pairs from the same cluster when computing average pairwise similarity.

A final parameter in addition to the linkage criterion, is the similarity measure applied to individual examples as used by the linkage criterion.

From the dendrogram in figure 3.1, several possible cuts are possible for retrieving clusters. One can perform cuts at a prespecified level of similarity. Performing cuts higher in the dendrogram would generate a smaller number of clusters with a higher number of words in each cluster, while a cut lower in the dendrogram would generate a higher number of clusters with fewer words in each cluster. For example, cutting at level 0.1 would only generate singleton clusters, since there are no agglomerations at or below this level. Cutting at level 0.8, on the other hand, would generate two clusters, corresponding to the two agglomerations between 0.6 and 0.8:

- *first, new, last, month, year, week, percent, trade, stock, share, price, rate, group, bank, market, company, government, n/a, newsroom, pct, u.s.*

- *expect, say, rise, end*

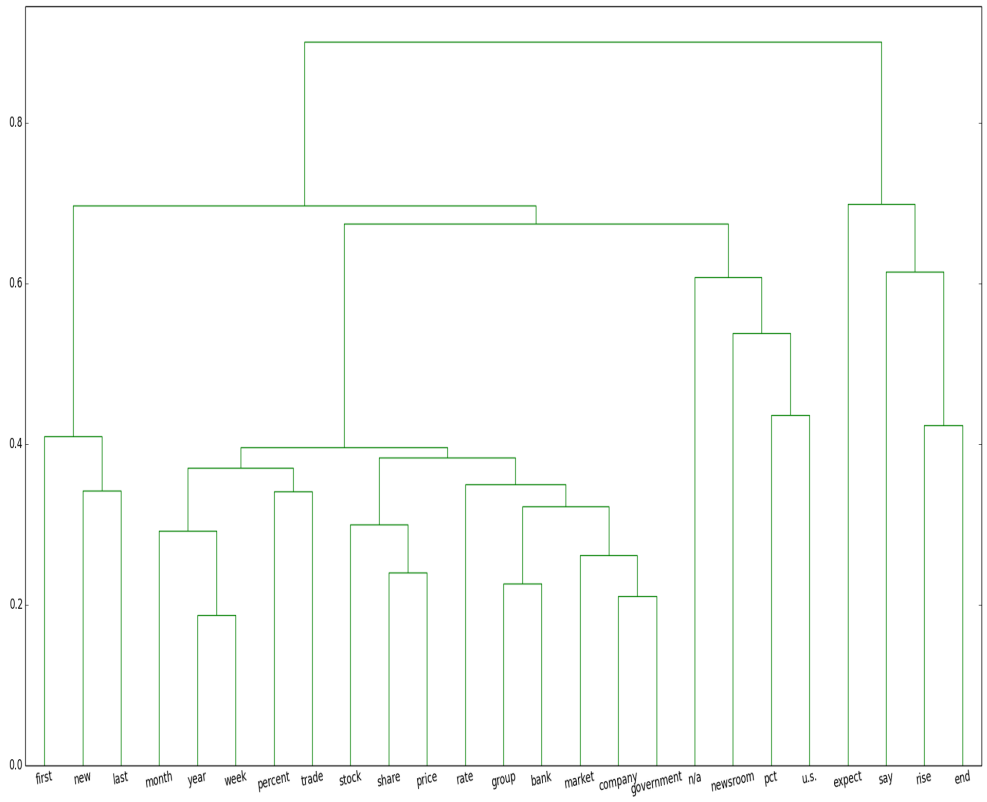


Figure 3.1: *A dendrogram representing the average-link clustering of the 25 most frequent words from the Reuters corpus. The clustering were performed with Python's Scipy clustering package, and using the dendrogram function for plotting the clustering. The y-axis shows the level of cluster similarity.*

### 3.2.2 Brown clustering algorithm

The Brown hierarchical clustering algorithm (Brown et al., 1992) is a bottom-up agglomerative clustering algorithm, where words are clustered based on their context. Being an effective and simple algorithm, it has been used in several natural language processing tasks, for instance dependency parsing (Koo et al., 2008; Candito and Seddah, 2010). In an $n$-gram language model, the probability of a word in a sequence of words is conditioned only on the last $n$-1 words. The *bigram* language model for instance, conditions a word's probability only on the previous term in the sequence (Manning et al., 2008). The probability of the word sequence $\mathrm{P}(w_1, w_2, w_3)$ is then found by calculating $\mathrm{P}(w_1)\mathrm{P}(w_2|w_1)\mathrm{P}(w_3|w_2)$. The main idea in the Brown algorithm is to merge the pair of clusters that causes *the smallest decrease in the likelihood of a text corpus*, according to a class-based bigram model defined on the word clusters. (Koo et al., 2008).

The Brown algorithm operates in the following way (Koo et al., 2008):

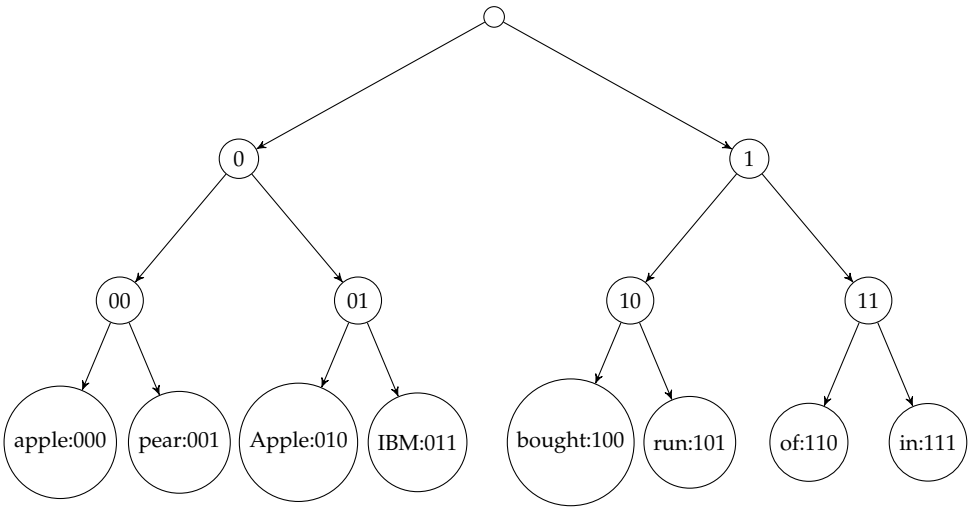


Figure 3.2: *A Brown cluster hierarchy, where each word is associated with a bit-string, corresponding the path traversed from the root node down to a leaf node. 0 indicates a left traversal and 1 indicates a right traversal.*

1. The input is a vocabulary of words to be clustered along with a corpus of text that include the words.
2. At first, each word is contained in it's own singleton cluster.
3. Then merge into a new cluster the two clusters having the smallest decrease in the likelihood of the input corpus, given a class-based bigram language model defined on the clusters.
4. The merging of two clusters in the step above is repeated, gradually constructing a binary tree of merges.
5. After the final merging operation is complete, a Brown word-cluster hierarchy is complete.

The Brown algorithm can then be seen as an agglomerative clustering, where cluster-similarity is defined on the basis of likelihood in the bigram language model. In the constructed binary tree representing the cluster hierarchy, each word is associated with a bit-string. The actual bit-string for a word is given by the path from the root of the binary tree, to the leaf node representing the word. As one traverses from the root node down to the leaf, the symbol *0* indicates a left traversal, and *1* indicates a right traversal. Figure 3.2 presented in (Koo et al., 2008), shows an example of a Brown cluster hierarchy. For instance, the bit-string *010* associated with the word *Apple*, is found by starting at the root node, traversing left (*0*), then right (*1*), and finally left (*0*), until the string *010* is constructed. Two words being close to each other, will then have similar bit-strings.

One can obtain a clustering from the Brown hierarchy by selecting the nodes at a certain depth in the tree. Before the cluster-hierarchy is cut, each word belongs to as many clusters as there are nodes between the word and the root node. From the hierarchy shown in figure 3.2, one can select the two nodes *0* and *1* at depth 1 starting from the root. The two clusters {*apple, pear, Apple, IBM*} and {*bought, run, of, in*} are then retrieved.

By extracting the prefix of a bit-string, clusterings can also be retrieved. The aforementioned clustering can then be retrieved by using only a 1-bit prefix: *0* for the first cluster and *1* for the second. The length of the bit-string can then vary in order to retrieve clusterings of different granularities (Koo et al., 2008). A longer bit-string would result in a higher number of more specific clusters, each having fewer words. Using all the bits in the bit-string would then result in only singleton clusters. The same strategy can of course be used for any hierarchical clustering, not just a tree produced using the Brown algorithm.

## 3.3 K-means clustering

The K-means algorithm generates a specified number of $k$ flat clusters, with no specific structure relating clusters to each other. This is an instance of hard clustering, where each object to be clustered belongs to exactly one cluster. Cluster centres are defined by the mean or *centroid* of the objects belonging to a cluster, so that the centroid represents the centre of mass of the cluster members. The centroids also play a similar role in *Rocchio classification*, an instance of supervised learning. In order to find the distance between objects, some way of measuring distance is needed. The most common distance measure used in K-means , is the *Euclidean distance*. The objective of the K-means algorithm is then to minimize the average squared Euclidean distance of the objects to be clustered from their cluster centres (Manning et al., 2008).

The K-means algorithm works in the following way (Marsland, 2009):

1. For initialization, set a value for $k$, corresponding to the number of clusters to generate. In the input space, $k$ random locations are selected. These so-called seeds are the initial cluster centers, and will have their locations updated through the iterations of the algorithm.
2. For each object to be clustered, it's distance to each of the $k$ cluster centers is computed. The object is then assigned to the cluster having the shortest distance between its cluster center and the object.
3. Each cluster center will then have its position updated by calculating the mean of all the objects in the cluster. The iteration then repeats from step 2 until some stopping criteria is met.

For terminating the algorithm, one of the following *stopping criteria* may be applied (Manning et al., 2008):

- The assignment of objects to clusters does not change between iterations.
- There is no change in the cluster centers between iterations.
- After a specified number of iterations are completed.

For the selection of the initial seeds in step 1, other approaches beyond picking random locations in the input space, include picking random objects from the data as initial cluster centers, or using pre-selected objects

as initial cluster centers. The non-deterministic behavior of the K-means algorithm due to random initialization, can in some cases impact on the resulting clustering. In section 6.1.5 we quantify this effect for our data sets. Another drawback with K-means is that the algorithm is susceptible to noisy data objects, so-called *outliers*. This is because of the mean average, central in K-means. An alternative to the mean average is to use the median instead. This is a more robust statistic, being less affected by outliers. The use of the median statistic is, however, more computationally expensive (Marsland, 2009).

### 3.3.1 Mini-batch K-means

An optimization of the regular K-means clustering algorithm (classic batch) has been presented in order to scale large data sets (Sculley, 2010). This optimization makes use of so-called mini-batches for reducing computation time. A mini-batch is a subset of the data, that for each training iteration is randomly sampled. There are two iteration steps in the mini-batch algorithm, performed until convergence or a specified number of iterations has been reached:

- Draw $n$ samples at random from the data to be clustered, creating a mini-batch. Assign each sample to the nearest centroid.
- Update the centroid for each sample in the mini-batch.

The classic batch K-means is expensive in terms of computation time for large sets of data, but the mini-batch variant, having low computation cost, has shown to give results being only slightly worse than the classic batch-variant for such large data sets (Sculley, 2010). For generating the clusters we use as a basis for our parsing experiments, we use the mini-batch variant of K-means implemented in scikit-learn (Pedregosa et al., 2011). This was far more efficient in terms of time and memory usage compared to the classic batch-variant of K-means . The following is a description of the various parameters used in that mini-batch implementation:

- **init:** method of initialization ('kmeans++' or 'random').
- **n-init:** number of random initializations that are tried.
- **max-iter:** maximum number of iterations over the dataset.
- **batch-size:** size of the mini-batches.

## 3.4 Hierarchical vs non-hierarchical clustering

In order to compare different approaches, some attributes of hierarchical and non-hierarchical clustering algorithms have been proposed (Manning and Schütze, 1999). In hierarchical clustering, there are no single best algorithm. They provide more information than flat clustering and are chosen for detailed data analysis. The are, however, less efficient than non-hierarchical clustering, since an n x n matrix of similarity coefficients needs to be computed.

Non-hierarchical algorithms, where K-means is regarded as the one of the simplest algorithms, is preferred when clustering large data sets and efficiency is important. This method of clustering provides often sufficient results, and should then be the first one to choose on new data sets.

In the experiments reported in chapter 6, we first decide on a vocabulary of words to cluster, where each word is represented as a feature vector. We then need to specify the size of the vocabulary, that is, how many words there are to be clustered. In addition, we also need to decide on the number of features to use for representing the words with feature vectors. These choices will affect *scalability*, the algorithm's ability to handle the input data. More specifically, we see this in terms of run time and memory usage of the clustering algorithm. For the huge vocabulary size of 50,000 words, the only clustering algorithm with a reasonable capability, was the mini-batch variant of K-means from scikit-learn's machine learning package, written in Python. We took advantage of its ability of representing the words to be clustered with sparse matrices, that is, storing only non-zero values. Unfortunately, sparse matrices are not implemented in the agglomerative clustering routines in the Scipy package for Python, where the use of dense matrices, storing both zero and non-zero values, are required. We used the agglomerative clustering functionality from this package in order to experiment with clusters generated from this algorithm compared to using K-means, as explained in chapter 6.

## 3.5 Evaluation of clusters

An *internal criterion* for assessing the quality of a clustering is to achieve high intracluster similarity, where objects in the same cluster are similar, and low intercluster similarity, where objects from different clusters are dissimilar. A good result on such an criterion for a clustering will, however, not mean that the clustering has a high degree of usefulness in a specific application. As an alternative to internal criteria, we may evaluate the clustering more directly in the specific application of interest (Manning et al., 2008).

For the purpose of this thesis, we generate clusters of words and use the word's cluster-label id in a cluster-based feature model with MaltParser to see how this improves statistical dependency parsing. In this case, we perform an *extrinsic evaluation* of the clusters, where the quality of clusters is measured in terms of how they affect parsing performance.

## 3.6 Summary

We have in this chapter explained two approaches of a machine learning technique, more specifically *hierarchical* and *non-hierarchical* clustering. Hierarchical clustering creates a hierarchy of clusters, having the form of a tree structure. The Brown hierarchical clustering algorithm is a bottom-up agglomerative clustering algorithm, where words are clustered based on their bigram context. The K-means algorithm is a general non-hierarchical

clustering algorithm that generates a specified number of $k$ flat clusters, with no specific structure relating clusters to each other. It can be used with any type of feature function and is not tied up to an $n$-gram model. An optimization of the K-means clustering algorithm, making use of mini-batches for reducing computation time has been presented in order to scale large data sets. The mini-batch K-means implementation in scikit-learn is the variant of K-means we will be using in our experiments presented in chapter 6, making use of a *sparse feature representation* for the words to be clustered. This means recording only values for the features that is non-zero for each word. A drawback with the implementation of agglomerative clustering in scikit-learn (which again builds on SciPy) is the lack of support for sparse-matrices. It requires the words to be clustered represented as *dense feature vectors*, recording values for all possible cluster features, both zero and non-zero. This will then cause poor scalability, since most of the feature values for a word is zero.

# Chapter 4

# Previous work

This chapter will give an overview of relevant previous work in the use of word clusters for improving statistical parsers. Previous work in this field have typically been using the *n*-gram-based Brown hierarchical clustering algorithm (Brown et al., 1992) we described in the previous chapter. We describe studies by Koo et al. (2008), Candito and Seddah (2010) and Øvrelid and Skjærholt (2012) who used clusters generated by the Brown algorithm. An alternative study we describe is that of Sagae and Gordon (2009) who instead of using Brown clusters, used parsed data for creating syntactically informed clusters. The emphasis will be put on describing the corpora used for clustering and parsing, the use of cluster-based features, and the parsing results for the aforementioned studies. A last addition to this chapter is an overview of the SANCL 2012 shared task in web parsing.

## 4.1 Supervised Dependency Parsing in English and Czech

A study describing the use of Brown clusters in dependency parsing, is presented by Koo et al. (2008). This work was inspired by Miller et al. (2004), who used word clusters in a discriminative learning approach of named-entity recognition. The work by Koo et al. (2008) is also similar to a Chinese dependency parsing approach by Wang et al. (2005) in the use of Brown clusters. However, Koo et al. (2008) focuses on discriminative learning instead of generative models. The word clusters were derived from a large unannotated corpus (Koo et al., 2008).

Koo et al. (2008) showed that a semi-supervised approach incorporating cluster-based features in a discriminative learner resulted in substantial performance gains over a baseline parser when performing a series of dependency parsing experiments on the Penn Treebank (Marcus et al., 1993) and Prague Dependency Treebank (Hajič, 1998) for English and Czech, respectively.

In the experiments of Koo et al. (2008), a baseline set of features is based on combinations of part-of-speech and words for the head and modifier for each dependency, as well as triples of part-of-speech tags for grandparent interactions, and sibling interactions. There are also bigram

features involved, which are based on word pairs. An example of a baseline feature template, is *ht, mt*. This indicates a class of features, containing a feature for each possible combination of the part-of-speech tag for the head and modifier part-of-speech tag (Koo et al., 2008).

The cluster-based feature set employed by Koo et al. (2008), uses information from the Brown cluster hierarchy. This set include all the features from the baseline set, in addition to features using word clusters. Given the Brown cluster hierarchy, clusterings with different granularity are generated by using the prefixes from the hierarchy, following the work of Miller et al. (2004). Koo et al. (2008) used two different types of word clusters (Koo et al., 2008):

- Bit-strings with 4 to 6 bits of prefixes length, replacing part-of-speech.
- Bit-strings of full length, substituting word forms. Koo et al. (2008) recovered a maximum of 1,000 different bit-strings, by limiting the Brown algorithm. The full bit-strings were then not equivalent to word forms.

The two cluster types were used for constructing new features, by using the baseline feature template structures (Koo et al., 2008).

Koo et al. (2008) also used hybrid features that involved features from both the baseline and cluster-based feature sets. For instance using bit-strings and part-of-speech for defining a feature template. Using these hybrid features contributed to better parsing performance, compared to using only cluster-based features. A possible reason for this, is a noisy or weak relevance between the Brown clusterings and syntax. Clusters would then come to better use when used in combination with part-of-speech or words. (Koo et al., 2008).

By performing dependency parsing experiments in both English and Czech, the parsing performance using cluster-based features was evaluated. Results were presented as parent-prediction accuracy, which is the percentage of tokens being attached to the correct head token. Parsers were trained with the *averaged perceptron* (Freund and Schapire, 1998; Collins, 2002), giving a trade off between fast training and performance (Koo et al., 2008).

For the English experiments, parsing was performed on the Penn Treebank (Marcus et al., 1993). As is standard, this was split into a training set (using sections 02-21), a development set (using section 22), and test sets (sections 0, 1, 23 and 24). Clusters for English words were derived from the BLLIP corpus (Charniak et al., 2000). Furthermore, Koo et al. (2008) made sure that the Penn Treebank sentences were excluded from the text used for clustering. In the experiments, eight different parsing configurations were tested, including all possible choices between baseline or cluster-based parsing (Koo et al., 2008).

Using the sibling interactions resulted in improved parsing accuracy, while adding the grandparent interactions gave even more improvement. Parsers with cluster-based features, also performed better than the baseline parsers, regardless of label usage (Koo et al., 2008).

Experiments in Czech were performed on the Prague Dependency Treebank 1.0 (Hajič, 1998), annotated with dependency structures. Koo et al. (2008) also used the predefined splits for training, development and test sets in the Treebank. Clusters of Czech words were derived from the Prague Dependency Treebank 1.0, having around 39 million words of text in newswire. The text used for clustering were disjoint from the text defined as training and test sets (Koo et al., 2008).

Parsing were also performed in the Czech with parsers using baseline and cluster-based features. Tuning of the feature sets were only performed for the features for Czech clusters. Apart from that, the feature sets were then based on the tuning for English parsing (Koo et al., 2008).

The Czech parsing results revealed the same trends as for English parsing. Parsers using cluster-based features performs better than parsers with baseline features, yielding statistically significant improvements in parsing accuracy (Koo et al., 2008).

Overall, the semi-supervised learning approach led to substantial improvements, using parsers informed with cluster-based information, as compared to baseline parsers, in both English and Czech dependency parsing experiments. Koo et al. (2008) also suggested some improvements. Since the Brown algorithm is based on a bigram language model, Koo et al. (2008) argues that a mismatch may occur between the lexical information provided by Brown clusters, and the lexical information in dependency parsing. A suggestion for future work would then be using clustering algorithms focusing on syntactic word behaviour. Another idea is to use a clustering algorithm in the algorithm for training. After a parser is trained, large amounts of unlabeled text could be parsed, and then use the parser for improving cluster quality (Koo et al., 2008).

## 4.2 Word clusters for parsing French

Candito and Seddah (2010), being inspired by (Koo et al., 2008), applied Brown clusters in statistical constituent parsing of French, by creating clusters over lemmas and part-of-speech pairs, using a raw corpus that was automatically part-of-speech tagged and lemmatized.

In all the experiments, parsing was performed on the French Treebank (Abeillé et al., 2003), having 350,931 tokens and 12,531 sentences from the *Le Monde* newspaper. The Treebank was partitioned into the first 1,235 sentences (10%) for testing, the next 1,235 sentences (10%) for development, and the remaining 9,881 sentences (80%) for training (Candito and Seddah, 2010). Parsing was performed with the Berkeley PCFG parser with latent annotations (Petrov et al., 2006), a *constituent* parser. The clusters were created from the L'Est Républicain corpus[1], containing 125 million words of journalistic content. This corpus was first tokenized and segmented into sentences (Candito and Seddah, 2010).

Candito and Seddah (2010) analysed the results with respect to word frequency and found improvements in performance for all strata; unseen

[1] `http://www.cnrtl.fr/corpus/estrepublicain/`

or rare words, as well as medium- to high-frequency words. Adding part-of-speech information to the lemmas also appeared beneficial, this would also depend on the quality of the tagger.

Suggestions for future work, was using other clustering algorithms making use of semantic or syntactic similarity. The clusters generated by the Brown algorithm makes use of bigrams, and this local information could cause noisy clusters (Candito and Seddah, 2010).

## 4.3 Syntactic word clustering for improving dependency parsing

As an alternative approach to using the n-gram-based Brown clusters, where words are clustered by *n*-gram word context, Sagae and Gordon (2009) followed a different approach by generating clusters of words, based on the words general syntactic contexts of appearance. Other clustering approaches may rely on *lexical context* for grouping words, based on both syntactic and semantic characteristics. Sagae and Gordon (2009), on the other hand, used *unlexicalized syntactic context*, meaning that words are clustered based only on their syntactic behaviour (Sagae and Gordon, 2009).

The goal of (Sagae and Gordon, 2009) was to achieve improved parsing performance of a predicate-argument dependency parser by using a corpus previously automatically annotated by using a phrase-structure parser. More specifically, given a large corpus of constituent parse trees generated by a slow but highly accurate constituent phrase structure tree parser (Charniak, 2000), these parse trees were used to generate syntactically derived clusters. These clusters were then used for improving the accuracy of a fast but less accurate dependency parser that outputs dependency graphs. The slower constituent parser was used to parse the data used for generating the syntactic word clusters, while the faster dependency parser used these clusters in order to improve its accuracy (Sagae and Gordon, 2009).

The clustering method used by Sagae and Gordon (2009), was hierarchical agglomerative clustering, where the distance between clusters was calculated using average-link clustering. Gordon and Swanson (2007) described the idea of using parse tree paths as features, as a basis of calculating syntactic similarity between words. The idea is that parse tree paths could be used as features in the description of the grammatical behaviour of words. By following the approach of Gordon and Swanson (2007), Sagae and Gordon (2009) also used frequency counts of parse tree paths for computing syntactic similarity during clustering. Given a sentence represented as a constituent parse tree, a parse tree path describes a tree transition from a terminal node to a different node in the tree. The path is represented as a sequence of arrows showing the direction av traversal and the part-of-speech tags that occurs along the path, such as $\uparrow$ VBD $\uparrow$ VP $\uparrow$ S $\downarrow$ NP, when travelling between two nodes in the parse tree (Sagae and Gordon, 2009).

For a word to be clustered, all possible parse tree paths beginning at a

word are identified. By normalizing the frequency counts of unique parse tree paths for the word, a feature vector for the word is created. This vector will then describe the location where the word appears in the set of parse trees (Sagae and Gordon, 2009).

Sagae and Gordon (2009) described two drawbacks of the representation of parse tree paths proposed by Gordon and Swanson (2007). One of them is the underspecification of path directionality by only using up and down arrow specification when describing paths. In order to avoid possible identical paths starting from different words in the tree, Sagae and Gordon (2009) expanded the original set of two identifiers $\uparrow$ and $\downarrow$, to also include direction of the transitions with, using the identifiers $\nwarrow$, $\searrow$, $\nearrow$ and $\swarrow$ (Sagae and Gordon, 2009). These could then be used in order to specify any up and down direction more fine-grained.

For generating clusters of words, Sagae and Gordon (2009) used hierarchical agglomerative clustering on the 5,000 most frequent words from the BLLIP (Charniak et al., 2000) Wall Street Jorunal (WSJ) corpus, having around 30 million words of WSJ news articles, parsed with the Charniak (Charniak, 2000) parser. Between each of the 5,000 words, the pairwise distance was calculated as the cosine distance between their feature vector representations. Since the words were clustered based on unlexicalized syntactic contexts, clusters would then reflect purely syntactic information, more than clusters derived from lexical context (Sagae and Gordon, 2009). The cluster labels would then in some sense be similar to part-of-speech tags, since they would indicate the syntactic contexts where the words appear. The baseline parsing model consisted of a core set of thirteen feature templates, in addition to features obtained by concatenating two of three core features (Sagae and Gordon, 2009).

For each of the hierarchical clusters obtained, unique cluster labels were assigned. These labels would then be used for generating additional *cluster-based features*, that the dependency parser would use to make decisions, based on word's syntactic profile. By using development data, different levels of cluster granularity was experimented with. For selecting a set of cluster labels to be used in the generation of features, the level of cluster granularity had to be specified, then using the set of cluster labels that resulted from slicing the dendrogram at that appropriate level (Sagae and Gordon, 2009). With a specified level of cluster granularity, the cluster-based features could then be defined.

The dependency parser was used to identify predicate-argument dependencies extracted from the HPSG Treebank, developed by Miyao et al. (2004). From this treebank, section 02-21 of the WSJ data was used as training, section 22 for development and section 23 for testing. Only information regarding predicate-argument structures was used, not making use of other information from HPSG. Evaluation of the parsing experiments were based on labeled precision and recall of predicate-argument dependency pairs (Sagae and Gordon, 2009).

First, a parsing model using only the baseline features was trained. These features was used on the basis of experiments performed on the development set. By applying the baseline model on the test set, the result

was a labeled precision of 88.7% and a recall at 88.2%. The development set was then used for investigating the effect of cluster sets with different levels of granularity. On the development set, the baseline model had a precision of 88.6% and a recall of 88.0%. Using a cluster-based model resulted in a statistically significant ($p < 0.005$) increase of 0.3 to 0.4 percentage points over the baseline model on precision and recall, when slicing the dendrogram to generate 50 to 100 cluster labels (Sagae and Gordon, 2009).

When increasing the number of cluster labels in steps of 100, (Sagae and Gordon, 2009) observed improvements in precision and recall when parsing with the cluster-informed parser up to the point of 600 cluster labels. When the dendrogram was cut to include 600 distinct cluster labels, the highest values of precision (89.5%) recall (89.0%) and F-score was observed (Sagae and Gordon, 2009).

In the vein of Koo et al. (2008), Sagae and Gordon (2009) also performed experiments with two sets of cluster labels simultaneously, having different levels of granularity. Using an additional set with fewer than 600 labels in combination with the set of 600 labels, did not affect precision and recall in any direction. When using finer grained clusters with more than 1,000 labels in combination with the 600 labels, however, improvements were observed. The highest scores of precision (90.1%) and recall (89.6%) was observed when using a combination 600 and 1,400 labels (Sagae and Gordon, 2009).

Sagae and Gordon (2009) were inspired in many ways by the work of Koo et al. (2008), who reported improved results on dependency parsing using word clusters derived from plain text with the Brown clustering algorithm. There is, however, a significant difference in the work of Sagae and Gordon (2009), who uses parsed data for generating syntactic word clusters. The experiments by Sagae and Gordon (2009) showed that the use of these syntactic word clusters based on word similarity, were effective for improving parsing accuracy in a transition-based dependency parser, contrary to (Koo et al., 2008) who used n-gram based Brown clusters from plain text (Sagae and Gordon, 2009).

## 4.4 Dependency parsing of web data

This section describes the experiments by Øvrelid and Skjærholt (2012) using features with information about cluster labels based on clusters generated by the Brown clustering algorithm, as well as using lemma information, to improve data-driven dependency parsing with MaltParser (version 1.4.1). Parsers trained on the Wall Street Journal (WSJ) are evaluated on a range of web texts in addition to the test section of WSJ. Clusters of words derived from unlabeled data has previously been shown to improve dependency parsing of English (Koo et al., 2008), in addition to using clusters derived from parsed data (Sagae and Gordon, 2009).

The WSJ portion of the Penn Treebank sections 2-23, were used in the experiments. More specifically, this included section 2-21 for training parsers and section 23 for evaluation of the parsers. Training

were performed on both the original LDC version and the version from the OntoNotes corpus release 4.0 (Weischedel et al., 2011), because of differences in tokenization. The OntoNotes corpus has also a range of treebanked web data from different sources, used in the experiments of Øvrelid and Skjærholt (2012) for evaluating parsers. The corpus has a total of around 23,000 sentences and 500,000 tokens, including punctuation. Furthermore, the OntoNotes data are provided split into six data sets that were parsed in their experiments (Øvrelid and Skjærholt, 2012):

- *p2.5_a2e* (Arabic translated to English with 16,000 tokens)
- *p2.5_c2e* (Chinese translated to English with 22,000 tokens)
- *a2e* (Arabic translated to English with 55,000 tokens)
- *c2e* (Chinese translated to English with 74,000 tokens)
- *eng* (General English web data with 71,500 tokens)
- *sel* (A set of sentences selected to improve sense coverage in the corpus, with 279,000 tokens)

In addition to the data sets mentioned above, twitter and football user forum data Foster et al. (2011) were also parsed in the experiments. These data contains a total of 1,000 sentences, split into test and development sets.

For *preprocessing* the data, the Stanford parser (version 2.0) with basic settings were applied. In this first step, the treebanked data were for each sentence converted into dependency graph representations from the Penn Treebank phrase-structure trees. After completion of converting the data to dependency representations, part-of-speech tagging with SVMTool (Giménez and Màrquez, 2004) were performed, using the pre-trained model for English. For the OntoNotes data, hyphens were converted into the HYPH part-of-speech tag. The data were then lemmatized with the WordNet lemmatizer from NLTK (Bird et al., 2009), where each token had its lemma written into the proper column in the CoNLL data format used for the dependency representations. This step was performed both on the gold tagged data and the automatically data, since the lemmatizer makes use of the token's part-of-speech tag. By using the cluster labels described by Turian et al. (2010), the data were also enriched with these cluster labels, generated by the Brown hierarchical clustering algorithm. The clusters consisted of words from the Reuters RCV1 corpus, containing English news stories (Øvrelid and Skjærholt, 2012).

Using cluster based features in the experiments, the number of clusters was either set to be 100, 320, 1000 or 3200 (Øvrelid and Skjærholt, 2012). In addition to using full-length cluster labels, prefixes of the labels were also used, having various lengths from 4 to 6 following the same vein as Koo et al. (2008).

For baseline parsing, the parse model described by Foster et al. (2011) were used in MaltParser with the stacklazy algorithm (described in section 2.2.2), and liblinear [2] for inducing classifiers. There are in additional three extended parse feature models employed (described in table 2.1), for experimentation with lexical features derived from clusters or lemmas

[2] `http://www.csie.ntu.edu.tw/ cjlin/liblinear/`

(Øvrelid and Skjærholt, 2012).

In the experiments, two pairs of baseline parsers were trained on section 02-21 on the WSJ corpus: One pair of parsers for the WSJ data with original tokenization with gold part-of-speech tags and automatically assigned part-of-speech tags, respectively, and the other pair for the WSJ data with the OntoNotes tokenization with the two types of part-of-speech tags, respectively. Parsing performance was reported as the *labeled accuracy score* (LAS), expressing the proportion of tokens having both head and dependency label correctly assigned during parsing. Statistical significance were assessed using Bikel's evaluation comparator (Øvrelid and Skjærholt, 2012).

Parsing results showed that parsers employing the additional cluster labels and lemma features, performed better when trained and parsed with automatically assigned part-of-speech tags, compared to using using gold standard tags. When parsing with cluster-based features, there was significant improvements over the baseline result for the football forum data (statistically significant at $p < 0.05$), while there were small non-significant improvements for the twitter data. Best results were achieved with parsing models using the form features in combination with the smaller cluster numbers at 100 and 320 (Øvrelid and Skjærholt, 2012).

Overall, the experiments by Øvrelid and Skjærholt (2012) had shown that lexical features derived from lemmas and clusters improved dependency parsing of web data. Furthermore, the use of cluster labels and lemmas gave better results when used with parsers trained and tested on data with automatically assigned part-of-speech tags, instead of using gold tags (Øvrelid and Skjærholt, 2012). For future work, Øvrelid and Skjærholt (2012) suggested similar experiments with data from other genres and domains, and also using other parsers and clustering algorithms (Øvrelid and Skjærholt, 2012).

## 4.5 The SANCL 2012 Shared task on web parsing

In the SANCL 2012 shared task on parsing English web-data hosted by the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL 2012) (Petrov and McDonald, 2012), participants were given the task of building the best possible parsing system, being able to handle noisy text commonly found on the web, and also being robust to changes in domain. From a total of 20 submissions, 12 were for dependency parsing and 8 for constituent parsing (Petrov and McDonald, 2012). We will focus on the dependency parsing track in this section.

The shared task provided both unlabeled and labeled data for the five different domains from the English Web Treebank (Bies et al., 2012), Weblogs, Emails, Answers, Newsgroups, and Reviews, covering various genres commonly found on the web. Contesters in the task were provided with sections 02-21 of the Wall Street Journal (WSJ) portion of OntoNotes 4.0, and five sets (Weblogs, Emails, Answers, Newsgroups and Reviews) of unlabeled sentences from the English Web Treebank for training.

For development, two sets of labeled sentences were provided, more specifically the Weblogs and Emails domains. For held-out evaluation of the parsing systems, contesters were provided with sections 23 of the WSJ portion of OntoNotes 4.0, in addition to raw sentences from the test portion of the Answers, Newsgroups and Reviews domains from the English Web Treebank. Official evaluation was, however, only done on the Answer, Newsgroups and Reviews domains, while evaluation results for the WSJ 23 section was used for comparing in-domain and out-of-domain parsing performance. (Petrov and McDonald, 2012). In Chapter 5, we give a more detailed explanation of these data sets, that we use in our experiments.

The parsing systems built by the contesters, should be able to robustly parse all the aforementioned domains, instead fo building several domain-specific systems. A requirement was that the parsing systems should only be trained on the WSJ 02-21 and the five unlabeled English Web Treebank sections. It was important that the development sets (Weblogs and Emails) were not to be used for training the parsers (Petrov and McDonald, 2012).

For evaluating the submitted dependency parsing system outputs, the CoNLL 2006 *eval.pl* script was used for reporting labeled attachment score (LAS) and unlabeled attachment score (UAS). This is the same script we use when evaluating the parsing results in the experiments described in chapter 6. For the 2012 shared task, this script was, however, modified in order to deal with the noisy part-of-speech tags often predicted on web data (Petrov and McDonald, 2012).

From the results in the shared task, it is clear that domain adaptation for parsing web data poses a challenge; results in dependency parsing showed accuracies exceeding 90% when parsing newswire text (WSJ), but the best scores for parsing web data was in the range of 81–85%. The Answers domain containing questions and imperatives, being furthest from WSJ in terms of syntactic structure, gave the lowest parsing accuracies with LAS that varied from 68.54% to 81.5%. It is expected that parsing even more distant domains, such as social media texts (Foster et al., 2011), would then result in even lower parsing accuracies (Petrov and McDonald, 2012). LAS for the submissions in the dependency track further showed scores that varied from 81.74% to 91.88% for WSJ, 74.41% to 85.85% for Newsgroups, and 70.17% to 83.86% for Reviews (Petrov and McDonald, 2012).

The highest scoring systems in the shared task did, however, involve some kind of system combination. Parsing results for constituency outputs converted to dependencies, did better on average than systems in the dependency track. The highest scoring dependency system was a combination of converted constituency parsers, and this is explained by the constituent parser's ability to perform better when parsing out-of-domain data (Petrov and McDonald, 2012). For the systems from the constituency track converted to dependencies, the LAS was from 86.56% to 91.37% for WSJ, 79.78% to 85.47% for Newsgroups, 78.47% to 84.19% for Reviews, and 76.50% to 81.71% for Answers (Petrov and McDonald, 2012).

A general observation was that better scores for WSJ parsing led to better web parsing. In order to resolve the challenges of domain adaptation, it was then suggested that one could focus on improving

parsing on the WSJ. On the other hand, this was also believed to be caused by the number of different combination systems submitted to the shared task. DCU-Paris13, being the best scoring team, submitted a dependency parsing system that had the best accuracy on parsing web text, but was ranked as number seven on the WSJ evaluation data (Petrov and McDonald, 2012).

In particular for the dependency track, an observation was also that better part-of-speech tagging was related to higher parsing accuracy. Unlike constituent parsers, dependency parsers are more dependent on the accuracy these tags, since they are used as input to the parser. It was then suggested that focusing on accurate part-of-speech tagging on web data would be an area of attention when performing dependency parsing on web data (Petrov and McDonald, 2012). The next section will explain one of the submissions in the shared task, a dependency parsing system enhanced with dependency based Brown clusters.

### 4.5.1 NAIST dependency parsing for the SANCL shared task

In the SANCL task, Hayashi et al. (2012) submitted a dependency parsing system, employing the shift-reduce algorithm. Hayashi et al. (2012) generated clusters using dependency n-gram information based on head/child information instead of using word n-gram information based on left/right context like Koo et al. (2008). The unlabeled SANCL data were parsed with the MST parser (Mcdonald and Pereira, 2006), followed by extracting head/child bigram dependencies being input to the Brown clustering algorithm (Hayashi et al., 2012). As a baseline parser, (Hayashi et al., 2012) trained the dependency parser on sections 02-21 of the OntoNotes WSJ data. During development on the Emails section, the cluster-informed parser gave a LAS at 73.2% against 73.1% for the baseline parser. On the Weblogs section, the cluster-informed parser gave a LAS at 81.7% against 81.5% for the baseline parser. Test results reported by Hayashi et al. (2012) shows a LAS on Answers at 73.54%, Newsgroups at 79.83%, and Reviews at 75.72%. The WSJ test section got a LAS at 87.95% (Hayashi et al., 2012). We report the results using data sets with automatically generated part-of-speech tags.

## 4.6 Summary

Some prior studies on using word clusters for improving ngram-based statistical parsers have been described in this chapter. Koo et al. (2008) used Brown clusters in their experiments with dependency parsing in English and Czech, showing substantial gains in parsing performance. Candito and Seddah (2010) used Brown clusters in statistical constituent parsing for French, creating clusters of lemmas and part-of-speech tagged lemmas. Koo et al. (2008) and Candito and Seddah (2010) performs parsing only on news text. Øvrelid and Skjærholt (2012) applied Brown clusters for improving dependency parsing of English web data using MaltParser,

where the use of cluster information proved more beneficial for parsing with automatically assigned part-of-speech tags. Sagae and Gordon (2009) used parsed data for creating syntactically informed clusters. Their goal was to improve the accuracy of a fast dependency parser by using a corpus which had previously been automatically annotated using a slower but more accurate constituent parser. In the 2012 SANCL shared task on parsing the web, contesters were provided both labeled and unlabeled data for five the different domains from the English Web Treebank, in addition to the WSJ portion of the OntoNotes corpus. The goal in the shared task was to build parsing systems being robust to domain changes and being able to handle noisy web text (Petrov and McDonald, 2012).

Instead of using the *n*-gram-based Brown clusters, we will in our experiments described in chapter 6 be using syntactically informed clusters. After applying a baseline dependency parser to unlabeled data, clusters are generated by the K-means algorithm. This can be related to the study by Sagae and Gordon (2009) who also used parsed data for generating clusters. However, we use only one parser in our experiments, avoiding the complexity of involving a second parser. We then train a parser using labeled data with information about the generated clusters, making our approach a simpler way of semi-supervised learning. We also perform a comparison experiment with the work by Øvrelid and Skjærholt (2012), who applied Brown clusters for improving dependency parsing using MaltParser, the same parsing system we use. The English Web Treebank provided in the 2012 shared task, in addition to the unlabeled data from the same domains, is also used in our experiments, where we use the unlabeled data for generating clusters, and the labeled data for evaluating a cluster-informed parser. Parsers are trained on the WSJ 02-21 sections, and we parse both in-domain (WSJ section 23) and out-of-domain (labeled EWT domains) data. The next chapter will describe the data sets used in our experiments in more detail.

# Chapter 5

# Corpora and preprocessing

In this chapter we present the corpora we use as a basis for generating the clusters of words, in addition to the corpora for parsing. We will first give an overview of these corpora, then explain the various preprocessing steps.

## 5.1 SANCL data sets

The shared task on parsing English web-data hosted by the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL 2012) (Petrov and McDonald, 2012) provided both unlabeled and labeled data for the five different domains from the English Web Treebank (Bies et al., 2012), Weblogs, Emails, Answers, Newsgroups, and Reviews, covering various genres commonly found on the web. To produce these data, a large set of sentences (more than 1 million sentences in most cases) was collected. In order to produce the labeled versions of these data, a smaller subset of the sentences were randomly sampled, then annotated with syntactic parse trees by annotators from the Linguistic Data Consortium (LDC). This labeled treebank was then converted to labeled dependencies, represented in the CoNLLdata format for the shared task, using version 2.0 of the Stanford converter. The labeled data were for each domain also divided into two halves, one for development and one for evaluation. Contesters in the task were provided with sections 02-21 of the Wall Street Journal (WSJ) portion of OntoNotes 4.0, and five sets (Weblogs, Emails, Answers, Newsgroups and Reviews) of unlabeled sentences from the EWT for training. Each of these unlabeled data sets contained from 27,000 to 2,000,000 sentences. For development, two sets of labeled sentences were provided, more specifically the Weblogs (1,016 parsed sentences) and Emails (2,450 parsed sentences) domains (Petrov and McDonald, 2012). For the purpose of this thesis, we will refer to the five EWT domains as the SANCL domains.

## 5.2 Corpora for clustering

In order to build a vocabulary storing the words to be clustered, we must first choose a corpus containing these words. We choose to experiment with generating clusters of words from the Reuters corpus volume 1 (Rose et al., 2002), and the five unlabeled SANCL domains. Table 5.1 presents the number of sentences and tokens for the corpora used for parsing.

| | Reuters | Weblogs | Emails | Answers | Newsgr | Reviews |
|---|---|---|---|---|---|---|
| Sents | 12,515,901 | 524,834 | 1,194,172 | 27,274 | 1,000,000 | 1,965,350 |
| Tokens | 217,635,636 | 10,356,138 | 17,046,119 | 424,292 | 18,424,049 | 29,288,947 |

Table 5.1: *The number of sentences and tokens in the corpora used for generating word clusters.*

### 5.2.1 Reuters Corpus Volume 1

Released by Reuters in 2000 for use in information retrieval and natural language-processing systems, the Reuters Corpus Volume 1 contains about 1 GB of text, more specifically 806,791 English newswire stories collected during a 1-year period, from 1996-08-20 to 1997-08-19. In total, there are about 100 million tokens, in average 222 tokens in each document. The number of total distinct terms is 391,523 (Manning et al., 2008).

The corpus is marked in XML, and we extract the text in the headline and text parts of all the newswire stories. Processing a newswire story is performed with the following procedure: Each sentence is extracted and stored on a separate line in a text file. This is done for all the newswire stories. Given all the sentences from the corpus on separate lines, we then for each sentence extract each token using NLTK's (Bird et al., 2009) word tokenizer function on the sentence as input. Each token is then written to a separate line. In addition, we add a blank line after the last token in each sentence, separating the sentences to conform to the CoNLLdata format.

With each token on separate lines, and each sentence separated with a blank line, the data is then ready for input to SVMTool (Giménez and Màrquez, 2004) for assigining a part-of-speech tag to each token. After completion of the part-of-speech tagging, the data is then saved in the CoNLLformat. The sentences are then parsed with MaltParser (version 1.7.2), using the baseline parser (This parser and the feature models it employs are described in chapter 2) trained on the WSJ 02-21 training section, giving each token values in the head and dependency columns. Now that the Reuters data in the CoNLLformat, it can be used for building a vocabulary for clustering, then subsequently used for generating clustering features.

### 5.2.2 Unlabeled SANCL domains

For generating clusters based on words from the unlabeled SANCL domains, we applied the same preprocessing procedure for each of the five domains as we did with the Reuters corpus for representing the sentences in CoNLLformat. Each unlabeled domain was initially represented in raw tex, one sentence on each line. The tokens were extracted, subsequently written on separate lines, with a blank line between sentences. Following part-of-speech tagging using SVMTool, each domain was then represented in the CoNLLformat, parsed with the baseline parser.

## 5.3 Corpora for parsing

For training and evaluating parsers, we use the Wall Street Journal (WSJ) portion of the OntoNotes corpus release 4.0 (Weischedel et al., 2011), and the labeled data from the English Web Treebank. Tables 5.2 and 5.3 presents the number of sentences and tokens for the corpora used for parsing.

| | Weblogs | Emails | Answers | Newsgroups | Reviews |
|---|---|---|---|---|---|
| Sents | 4060 | 4,900 | 6,976 | 4,782 | 7,627 |
| Tokens | 88,762 | 57,807 | 108,006 | 86098 | 111,182 |

Table 5.2: *The table presents the number of sentences and tokens in the labeled SANCL web data used for parsing.*

| | WSJ 02-21 | WSJ 23 | WSJ 22 |
|---|---|---|---|
| Sents | 30,060 | 1,640 | 1,336 |
| Tokens | 731,678 | 39,590 | 32,092 |

Table 5.3: *The table presents the number of sentences and tokens in the WSJ sections where section 02-21 is for training, section 22 for development and section 23 for testing .*

### 5.3.1 OntoNotes Wall Street Journal

For the purpose of the 2012 shared task, the WSJ portion of the OntoNotes corpus (release 4.0) was used. The difference between this and the Penn Treebank (Marcus et al., 1993) WSJ version, is a difference in certain aspects, such as tokenization and noun-phrase bracketing. Since the OntoNotes version share annotation standards with the English Web Treebank, the OntoNotes version was then chosen for the shared task and this thesis.

The data provided for the shared task contains the OntoNotes WSJ sections for training (sections 02-21), development testing (section 22), and held-out testing (section 23). These data are provided tokenized and

converted to the CoNLLformat using the Stanford dependencies. We do, however, replace the part-of-speech tags with automatic generated tags, using SVMTool for all the three aforementioned sections.

### 5.3.2 Labeled SANCL domains

Each of the five labeled SANCL domains are provided pre-converted to the CoNLLformat, with development and evaluation splits. We do, however, merge these two splits into one data set for each domain. Like in the set-up of the shared task, the Weblogs and Emails domains were used for development testing, and the Answers, Newsgroups, and Reviews domains were used for held-out testing.

### 5.3.3 Eng and Sel

Finally, we will perform experiments comparing our results with the results from (Øvrelid and Skjærholt, 2012). This includes using the OntoNotes WSJ, with sections 02-21 for training, and section 23 for testing. The experiments performed by Øvrelid and Skjærholt (2012) also included additional data from the OntoNotes corpus. Among these are general English web data (eng) with 71,500 tokens, and sentences selected to improve sense coverage in the corpus (sel), with 279,000 tokens (Øvrelid and Skjærholt, 2012).

The aforementioned data sets are provided as constituency parse trees and use the Stanford parser with the basic settings (version 2.0) for converting these data to the CoNLLdata format. The data is in addition part-of-speech tagged using SVMTool with the pre-trained model for English based on the Wall Street journal corpus (English WSJ). We perform the comparison experiment using data both with gold standard part-of-speech tags, and automatically generated tags.

## 5.4 Gold standard part-of-speech tags vs automatically assigned part-of-speech tags

In a preliminary round of experiments, we investigate the effect of replacing the gold standard part-of-speech tags in the Wall Street Journal sections, with automatically generated tags from SVMTool (Giménez and Màrquez, 2004). The input to SVMTool for tagging is a file with the tokens on each line, and each sentence separated with a blank line. We use SVMTool version 1.3 and the pre-trained English WSJ model. The tagging strategy used, is a combination of left-to-right and right-to-left tagging, as this has proven to give significant improved results, instead of using only one direction. This will, however, make the tagger twice as slow.

These experiments used the baseline MaltParser without cluster information and using only default parameter settings with a C-value at 0.1 for the SVM. We trained two versions of the parser on WSJ sections 02–21 (from OntoNotes/SANCL) using (1) the gold part-of-speech tags provided

in the treebank and (2) replacing these with tags automatically predicted by SVMTool. We then applied the parsers to WSJ 22 and 23, for both parsers using SVMTool tags during testing.

| Parse sections | GS pos-tags | AT pos-tags |
|---|---|---|
| Ont WSJ 22 | 81.54 | **84.88** |
| Ont WSJ 23 | 81.88 | **84.79** |

Table 5.4: *The effect on LAS for training on gold vs. predicted part-of-speech tags.*

The results shown in table 5.3 reveal that there is a clear advantage to training on predicted tags (all differences are statistically significant at $\alpha = 0.05$). For all parsing results reported elsewhere in this thesis, automatically predicted part-of-speech tags are used in both training and testing.

## 5.5 Preprocessing the data

### 5.5.1 Automatic part-of-speech tagging

We have seen that using automatically assigned part-of-speech tags resulted in a better parsing performance compared to using gold standard part-of-speech tags. For all the corpora used for clustering, i.e., the Reuters and the five unlabeled SANCL domains, we assign for each token a part-of-speech tag generated using SVMTool. This pre-pocessing step is also performed on all the corpora used for parsing, which is the OntoNotes WSJ sections 02-21, 22, 23, and the five labeled SANCL domains.

The first step is to extract each token from the corpus to be processed, then writing each token on a separate line, with a blank line between sentences. This is saved as a text file, then representing the input to the tagger.

For the purpose of assigning automatically generated part-of-speech tags to each token, we use SVMTool (version 1.3) with the following settings:

- Tagging direction -S = LRL (Left Right Left)
- Strategy -T = 0

The output from the tagger is a file where each line corresponds to a pair which is a token and associated part-of-speech tag. The gold standard tags are now replaced in the CoNLLfile data.

### 5.5.2 Lemmatization

An important decision is that we use the lemmatized form of the words for the vocabulary to be clustered. For instance, the word forms *looking* and *looked* will both be stored as the lemma *look* in the vocabulary. For extracting

the lemmas for each word form, we use NLTK's WordNet lemmatizer, writing the lemmatized word form into the third column of the input CoNLLdata set for each token. This step is executed before the vocabulary construction begins.

# Chapter 6

# Experiments

In this chapter we will present the various experiments performed. We will first describe some general assumptions regarding the experimental process. This includes the data sets for clustering and parsing, a general description of the experiment process, including extracting a vocabulary of the lemmatized words to be clustered, extracting clustering features, clustering of the lemmas, parsing with MaltParser and evaluating the parse results. This will be followed by a more specific description of each experiment, presenting the parsing results, cluster features applied, data sets, various parameters, and an analysis of the results. At the end of the chapter, we show some examples of clusters.

## 6.1 Experimental setup

### 6.1.1 Model tuning and development

Each experiment is divided into two main parts: *development* and *held-out testing*. During the development phase, we use a part of the data for experimenting with various parameter values, in order to find an optimal configuration of parameters. During the held-out testing, the best configuration of parameters found during development is then applied on the test data sections for a final evaluation of the optimal parameter values.

Clusters of lemmas are generated from the Reuters and unlabeled SANCL domains (see section 5.2). We experiment with generating clusters from each of the five SANCL domains individually, and all the five domains concatenated into a single data set.

For parsing, we use the Wall Street Journal Ontonotes data, split into sections 02-21 for training the parser in all the experiments, section 22 for development and section 23 for held-out testing. With the labeled SANCL data sets, we use the Weblogs and Emails sections for development, and the Answers, Newsgroups and Reviews sections for held-out testing.

For each clustering data set, we extracted a vocabulary of the 50,000 most frequent lemmas for clustering with the K-means algorithm. For comparison, we also generated clusters from a smaller vocabulary of 5,000 and 10,000 lemmas and performed parsing experiments described

in section 6.8. The standard batch variant of K-means was not able to reasonably process 50,000 lemmas in combination with the high number of features used in our experiments, typical at around 300,000: A single run of the algorithm took approximately 24 hours and a memory usage at approximately 200 gigabytes. We then decided to use the mini-batch variant of the algorithm instead (see section 3.3.1), where a single run took approximately 1 hour and about half the memory usage, compared to the standard K-means variant. The clustering sessions were executed on *sh.titan.uio.no*, a server for handling demanding computational tasks for the NLP community at the University of Oslo. The attempt of generating more clusters than 100, resulted in clusters with a high amount of singleton clusters, at least 30-40% in most cases. In addition to a tendency of returning a number of clusters less than the number specified, there was also memory constraints using these larger cluster numbers. For the purpose of this thesis, we refer to the use of the mini-batch algorithm simply as K-means.

During development on the Weblogs, Email and WSJ OntoNotes 22 sections, we tested a range of different C-values for training parsers in MaltParser. This *C* penalty parameter is used in the support vector machines (SVM) classification method, governing the trade-off between training error and decision margin. Soft margin classification is an extension to the SVM, allowing the decision margin to make classification mistakes. The *C* parameter works as a regularization term, for controlling overfitting (Manning et al., 2008): A high C-value provides a higher penalty cost of misclassifying objects, enforcing a more precise model with a smaller margin and having fewer training errors. A lower C-value gives a larger margin, increasing the number training errors and may cause underfitting. The C-value will have a large impact on the resulting classification model being trained, and in our case, the parser performance.

For each parser configuration (i.e., each combination of cluster features, cluster data set, cluster set size $k$, and parser feature model), we experimentally tuned the C-value on the development sets. The tables in the development sections present the labeled accuracy score of the best parser configuration. At first we tested C-values in the interval [$2^{-7}$, $2^{-6}$, ... $2^{6}$, $2^{7}$], doubling the value in each step. After inspection of the results, it became clear that the best performance was typically seen at 0.0625. A more fine-grained search was then run, testing C-valuesusing increments of 0.015, starting at 0.0625 and testing values both upwards and downwards. The development parts of the experiments presents the best parsing results when training the parser with the various C-values. The best C-value for each configuration was then re-used when applying the models in held-out-testing. Before turning to the parsing experiments, we will describe the clustering process in more detail, starting with the two different sets of features we defined for the lemmas to be clustered.

### 6.1.2 Clustering

#### Dependency cluster features

We start by describing a set of features we will refer to as *dependency* features, applying all the following 17 feature functions in the respective experiments. The following gives a brief description of these features.

For the target lemma to be clustered, there are two features:

- Part-of-speech-tag for the target
- Dependency label from the target to the target's head.

Three features are based on information from the head word of the target:

- Part-of-speech-tag for the head
- Dependency label for the head
- Lemmatized form of the head

The siblings of a target are defined as the tokens having the same head word as the target. Out of all the siblings, the leftmost sibling is then defined as the token most far to the left in the sentence, having the same head as the target. The righmost sibling is similary defined as the token most far on the right, having the same head as the target. Given each of these two types of tokens, there are three features for the target:

- Part-of-speech-tag for the sibling
- Dependency label for the sibling
- Lemmatized form of the sibling

For all the tokens having the target as the head word, we call these dependents of the target. Similar as with the siblings, the rightmost and leftmost dependents are then defined as the tokens most far to the right or left in the sentence, having the target as their head word. Given each of these two types of tokens, there are three features for the target:

- Part-of-speech-tag of the dependent
- Dependency label for the dependent
- Lemmatized form of the dependent

A lemma with the *dependency* features is shown in figure 6.1.

```
label_target=nn : 61
pos_head=NN : 52
label_head=pobj : 29
rightmost_sibling_pos=NN : 7
leftmost_sibling_pos=DT : 11
leftmost_sibling_lemma=an : 2
lemma_head=decline : 3
rightmost_sibling_label=punct : 39
leftmost_sibling_label=punct : 1
label_target=dep : 2
lemma_head=rise : 4
rightmost_dependent_pos=IN : 35
label_head=nsubj : 5
leftmost_dependent_pos=NNP : 7
rightmost_dependent_label=punct : 9
label_target=nsubj : 48
pos_head=VBD : 28
label_head=root : 36
leftmost_dependent_label=nn : 16
rightmost_dependent_label=nn : 12
```

Figure 6.1: *Example of dependency clustering features for the word 'production', where each feature is mapped to its frequency for this word.*

**Path-to-root-pos cluster feature**

We also experiment with a second type of feature set: A *path-to-root-pos* feature is a sequence of dependency labels, more specifically the *shortest path* traversing from the target word to be clustered, following dependency labels to the root word. The root's part-of-speech is concatenated to the end of this path. Considering all the possible paths from a target to the root, we find the shortest of these paths, using a Python implementation of Dijkstra's algorithm[1]. An example of a lemma with the *path-to-root-pos* clustering features is shown in figure 6.2.

[1] `http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/117228 from priodict import priorityDictionary`

```
nsubj–UP–VBD : 72
dobj–UP–VBD : 4
appos–UP–nsubj–UP–VBD : 9
conj–UP–appos–UP–nsubj–UP–VBD : 1
dobj–UP–ccomp–UP–VBD : 2
pobj–UP–prep–UP–VBD : 1
nsubj–UP–JJ : 5
rcmod–UP–nsubj–UP–VBN : 1
nsubj–UP–VBG : 6
nsubj–UP–conj–UP–VBN : 1
nsubj–UP–ccomp–UP–VBP : 3
nsubj–UP–VBP : 34
conj–UP–nsubj–UP–VBP : 2
nsubj–UP–VB : 4
xcomp–UP–VB : 1
pobj–UP–prep–UP–appos–UP–nsubj–UP–VBD : 1
nsubj–UP–ccomp–UP–VBD : 10
nsubj–UP–conj–UP–ccomp–UP–VBD : 1
iobj–UP–VBD : 2
nsubj–UP–parataxis–UP–VBD : 1
nsubj–UP–VBZ : 2
pobj–UP–prep–UP–nsubj–UP–VBZ : 1
```

Figure 6.2: *Example of the path-to-root clustering feature for the word 'trader'.*

The occurrence of *UP* in the path, means following a dependency relation from a dependent to its head (against the arrow), while *DOWN* means from the head to the dependent (same direction as the arrow). For instance, the feature *appos-UP-nsubj-UP-VBD : 9* is a feature which is a path following the *appos* dependency relation starting from the dependent *trader* to a head word, then following the *nsubj* relation to the root word (again a head), which is a verb. This feature occurs a total of nine times for *overture.*

### 6.1.3 Building a vocabulary

In order to build a vocabulary for clustering, we need to decide on a corpus for extracting the words. We experiment with building vocabularies from the Reuters corpus, as well as from the unlabeled SANCL data sets, both from each SANCL domain individually and all domains concatenated into one data set. The input for vocabulary construction is the CoNLLformat representation of these data sets, where each line corresponding to a token is read.

We do not consider all the words for populating the vocabulary. Rather, we use only verbs, nouns and adjectives. More spesifically, we filter out words not having a part-of-speech tag which is a noun (NN, NNS, NNP, NNPS), verb (VB, VBD, VBG, VBN, VBP, VBZ) or an adjective (JJ, JJR, JJZ).

An important decision here is that we use the lemmatized form of the

word for the vocabulary to be clustered. For instance, the word forms *looking* and *looked* will both be stored as the lemma *look* in the vocabulary. For extracting the lemmas for each word form, we use a script from (Øvrelid, Skjærholt) for writing the lemmatized form into the third column of the input CoNLLdata set for each token. This step is executed before the vocabulary construction begins.

Using the frequency distribution from Python's NLTK package as a data structure, the lower-cased lemma of each word form for the nouns, verbs and adjectives is read from the input CoNLL-formatted data and stored in the frequency distribution, with its frequency updated. Next, we decide the size n of the vocabulary, corresponding to the n most frequent lemmas in the frequency distribution. The vocabulary returned is then the n most frequent lemmas in the frequency distribution, each associated with a unique id. In addition, each lemma and its frequency is also returned.

Table 6.1 shows the 30 most frequent lemmas from the Reuters corpus that we use for building a vocabulary to be clustered, and table 6.2 shows the 30 most frequent lemmas from the five unlabeled SANCL domains concatenated into one data set that we also use for clustering.

**The 30 most frequent lemmas from the Reuters corpus**

| Lemma | Total frequency |
|---|---|
| say | 2738565 |
| percent | 853060 |
| year | 642793 |
| market | 563377 |
| share | 518910 |
| company | 470728 |
| bank | 470673 |
| new | 455009 |
| price | 400943 |
| government | 341911 |
| last | 330378 |
| u.s. | 327267 |
| rate | 307201 |
| trade | 291514 |
| rise | 290662 |
| week | 287309 |
| expect | 283436 |
| stock | 278918 |
| end | 269506 |
| first | 259975 |
| newsroom | 257948 |
| n/a | 251748 |
| group | 251332 |
| pct | 248335 |
| month | 247327 |
| net | 239117 |
| sale | 234036 |
| report | 232867 |
| make | 230879 |
| official | 230752 |

Table 6.1: *The 30 most frequent lemmas from the Reuters corpus with their total frequencies.*

**The 30 most frequent lemmas from the All SANCL corpus**

| Lemma | Total frequency |
|---|---|
| get | 235747 |
| go | 218398 |
| time | 171905 |
| great | 168381 |
| make | 161527 |
| service | 156762 |
| body | 143438 |
| say | 141492 |
| work | 137018 |
| good | 136264 |
| know | 132800 |
| place | 126048 |
| take | 125121 |
| 's | 123919 |
| need | 111510 |
| use | 104835 |
| call | 104498 |
| new | 100823 |
| people | 99094 |
| want | 96882 |
| year | 96808 |
| look | 93615 |
| business | 93450 |
| come | 93345 |
| day | 92932 |
| think | 91138 |
| see | 90987 |
| food | 89345 |
| best | 88257 |
| give | 86212 |

Table 6.2: *The 30 most frequent lemmas from the five SANCL domains concatenated with their total frequencies.*

### 6.1.4 Constructing the feature matrix

Given a vocabulary of lemmas to be clustered, we need to extract features for each of the lemmas. The set of features for each lemma is represented as a feature vector, with feature ids indexing the corresponding frequency count for the feature.

For extracting the features, the CoNLL data from where the vocabulary was built, is read sentence by sentence, line by line. For each occurrence of a lemma from the dictionary, the selected feature functions are applied, adding features to the lemma's feature vector, or updating its frequency if

a feature is already present.

The use of a Python dictionary for storing a feature's lemma is convenient, since only the actual features and their corresponding frequencies are stored, instead of using full vectors with a frequency entry for each possible feature in a fixed position. The latter would result in feature vectors storing a huge amount of zeros, since many features do not occur for a given lemma. With the dictionary representations of the feature matrix, only non-zero values are then stored, as the high-dimensional is very sparse.

After the CoNLLdata is read and features have been added, a full feature matrix has been constructed, storing for each lemma a feature vector consisting of the feature and frequency mappings found. We also keep track of the total count for each feature that was found.

With the full feature matrix, features with total counts below a predefined feature threshold are thrown out, resulting in a reduced feature matrix. The motivation for this is to avoid feature matrices becoming too large, particularly if the size of the vocabulary is large, making the matrices unsuitable as input to a clustering algorithm. Also, features with a total count of one, will occur only for one lemma, and this information does not contribute for generating clusters. We therefore set the feature threshold to a minimum of two.

### 6.1.5 Clustering process

With a feature matrix in place, we are ready to generate clusters of lemmas, where lemmas within a cluster are as similar as possible to each other, and as dissimilar as possible to lemmas in other clusters. We also apply length normalization to the vectors before input to the clustering algorithm.

With the K-means clustering algorithm, repeated runs with the same parameter values gave, however, some differences in the clusters. For the most part, the same lemmas appear in the same clusters, but there are some differences in how the lemmas are distributed throughout the clusters. A complication with respect to assessing the effect of K-means clustering is this non-deterministic behaviour of the algorithm, due to the random initialization of the seeds. In order to see how this affected the parsing results, we generated 100 clusters 10 times, using the same parameter values as input to the K-means algorithm. The vocabulary consisted of the 50,000 most frequent lemmas from the Reuters corpus, and the clusters were based on all the *dependency* features. For parsing we used the *Form_all* model and a C-value at 0.0625, parsing the OntoNotes WSJ 22 development section. The parameter values for K-means were: init='k-means++', batch_size=10000, n_init, and max_iter=150. Table 6.3 presents the basic statistics.

| Ten repeated clustering and parsing runs | |
|---|---|
| Min | 86.64 |
| Max | 86.89 |
| Mean | 86.78 |
| Median | 86.81 |
| Standard Deviation | 0.09476 |
| Variance | 0.008979 |

Table 6.3: *The basic statistics of LAS, after running K-means ten times with the same vocabulary and parameters.*

### 6.1.6 Parsing and evaluating

With the CoNLL files annotated with cluster ids for the lemmas that was clustered, we perform parsing experiments using MaltParser (version 1.7.2). For training, we use the OntoNotes WSJ sections 02-21 annotated with cluster ids. In addition, we also specify the parse model to use, and the C-value. The first step is to create a parsing model by running MaltParser in *learning mode*, as seen in figure 6.3:

```
java -Xmx6000m -jar maltparser-1.7.2.jar
-c baseline-model
-if my_conllx.xml -F lilja_features/baseline.xml
-i ontonotes-wsj-train-10cl.conll -m learn
-a stacklazy -l liblinear -d CPOSTAG -s Stack[0]
-T 1000 -grl top
-llo -s_4_-c_.0625 -v debug > LOG2> LOG-ERR2
```

Figure 6.3: *Command-line example for training the parser in MaltParser. By running this command, MaltParser creates a parsing model named baseline-model.mco, which is a Single Malt configuration file, based on the data in the training file ontonotes-wsj-train-10cl.conll.*

Using the parsing model created during learning, the next step is to parse a set of sentences in the CoNLL data format, by running MaltParser in *parsing mode*, as seen in figure 6.4:

The parsing score is evaluated by comparing it to a gold standard, more specifically comparing the values from the head and dependency label columns from the parse output, to the values in the corresponding head and label columns in the gold standard. For this purpose, we use the perl evaluation script[2] of the CoNLL-X shared task on multi-lingual dependency parsing, reporting:

- *Labeled attachment score (LAS)*: fraction of tokens having both head and dependency label correct when comparing the parse result to the gold standard.

[2] `http://ilk.uvt.nl/conll/software.html`

```
java -jar maltparser-1.7.2.jar
-c baseline-model
-if my_conllx.xml -i ontonotes-wsj-dev-10cl.conll
-o parsed-ontonotes-wsj-dev-10cl-baseline-result.conll
-m parse
```

Figure 6.4: *Example of parsing in MaltParser. This command will parse the sentences in ontonotes-wsj-dev-10cl.conll. The output parsed-ontonotes-wsj-dev-10cl-baseline-result.conll is then a CoNLLfile consisting of the sentences from the file input to parsing, only with new values in the head and deprel columns.*

- *Unlabeled attachment score*: fraction of tokens having only correct head.
- *Label accurracy score*: fraction of tokens with correct label.

In our experiments, we only report the *Labeled attachment score* (LAS). For assessing the statistical significance between two parsing scores, we use Dan Bikel's Randomized Parsing Evaluation Comparator, with $p \leq \alpha = 0.05$ are considered statistically significant.

We have now explained the general steps in the experimental process, and will in the next sections explain the actual parsing experiments performed. We will use different corpora for generating clusters of words, as well as parse a number of different data sets. The first series of experiments are based on clusters of words from the Reuters corpus, where we experiment with both the *path-to-root-pos* clustering feature and the *dependency* features. For each of these, we divide the experiments into a development session for finding and optimal configuration of parameter values from the development, and a held-out session for applying the best configuration on the test data.

The second series of experiments are based on in-domain clusters: For each of the five unlabeled SANCL data sets we generate clusters and perform parsing experiments on the corresponding labeled SANCL data sets, e.g generate clusters from the unlabeled Reviews section and parse the labeled Reviews section annotated with clusters. Experiments are performed based on clusters generated using the *path-to-root-pos* and *dependency* features separately, as we did in the Reuters experiments.

This is followed by an experiment where we also generate clusters based based on the *path-to-root-pos* and *dependency* features on all the five unlabeled SANCL domains concatenated into one data set, and perform parsing on each of the five labeled SANCL domains separately annotated with these cluster labels.

We also perform an experiment where we compare our results with those of (Øvrelid and Skjærholt, 2012), who used Brown clusters in their experiments when parsing web data from various sources.

## 6.2 Reuters clusters and the path-to-root-pos clustering feature

In this experiment we use the *path-to-root-pos* clustering feature, generating a total of 5,219,593 features from the Reuters corpus. With a feature frequency cut-off at 10, there are 293,558 features used for the lemmas to be clustered.

### 6.2.1 Development

On repeated runs with 'random' as the method of initialization with 100 clusters, the result was a number of clusters between 70 and 75. We therefore chose 'k-means++' as the method of initialization since that gave the specified number when attempting to generate 100 clusters. Using a number of clusters higher than 100, failed due to memory constraints. For the experiments in this section, the following parameter values were used for the K-means clustering algorithm: init = 'k-means++'; n-init = 5; max-iter = 150; and batch-size = 1000.

Tables 6.4, 6.5, and 6.6 presents the development results on the WSJ 22, Weblogs and Emails sections, respectively.

**WSJ 22 with Reuters clusters from path-features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 86.92 | 86.81 | 86.87 |
| *Form_simple* | 87.01 | 86.97 | 87.00 |
| *Form_all* | 87.01 | 87.00 | **87.06** |

Table 6.4: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on on the WSJ 22 development section using Reuters clusters based on the path-to-root-pos clustering feature. The baseline result was 86.72.*

**Weblogs with Reuters clusters and path-features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 80.05 | 80.06 | 80.20 |
| *Form_simple* | 80.23 | 80.17 | 80.32 |
| *Form_all* | **80.37** | 80.30 | 80.32 |

Table 6.5: *Best LAS for each combination of cluster number and MaltParser feature model and SVM parameter C on the Weblogs development section using Reuters clusters based on the path-to-root-pos clustering feature. The baseline result was 80.00.*

**Emails with Reuters clusters and path-features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 72.86 | 72.90 | 72.87 |
| *Form_simple* | 72.95 | 72.99 | 72.99 |
| *Form_all* | 72.89 | 72.99 | **73.05** |

Table 6.6: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on the Emails development section using Reuters clusters based on the path-to-root-pos clustering feature. The baseline result was 72.85.*

### 6.2.2 Held out testing

During development on both WSJ 22 and the SANCL Emails data, using Reuters clusters based on the path-to-root-pos features, the optimal configuration of parameters was the *Form_all* model, 100 clusters, and a C-value of 0.0625. On the Weblogs data, however, the best configuration was the *Form_all* model, 10 clusters, and a C-value of 0.0475. The LAS for 10 and 100 clusters on the Weblogs section with the *Form_all* model differs by only 0.05, however, which is not statistically significant. Overall, we conclude that the configuration with 100 clusters performed best during development, since it performed best in two out of three cases. We will then use the following configuration of parameter values for parsing during held-out testing on the WSJ 23, Answers, Newsgroups and Reviews test sections:

- Parsing model = *Form_all*
- Number of clusters = 100
- C = 0.0625

The results for the held-out testing are presented in table 6.7.

**Held out testing with Reuters clusters and path-root-pos feature**

| | WSJ 23 | Answers | Newsgroups | Reviews |
|---|---|---|---|---|
| Baseline | 86.88 | 73.10 | 76.13 | 75.01 |
| Clusters | **87.02** | **73.47** | **76.79** | **75.40** |
| Difference | +0.14 | +0.37 | +0.66 | +0.39 |
| Significant | no | yes | yes | yes |

Table 6.7: *Held out testing LAS on WSJ 23 and the labeled SANCL Answers, Newsgroups and Reviews sections using Reuters clusters based on the path-to-root-pos clustering feature. The configurations of parameters was the Form_all model, 100 clusters, and a C-value at 0.0625*

When applying these values during held-out testing on WSJ 23, there was an improvement of 0.14 percentage points over the baseline model.

This difference is not statistically significant. For the three labeled SANCL sections, Answers, Newsgroups and Reviews, the improvement over the baseline model was larger. The Answers section improved by 0.37 percentage points, Newsgroups by 0.66 percentage points and the Reviews section by 0.39 percentage points. All of these three results were statistically significant.

The best improvement of 0.66 percentage points was found on the Newgroups section. WSJ section 23 is the section having the smallest amount of tokens and sentences out of the four held-out test sections, with 39,590 sentences compared to 86,098 sentences in the Newsgroups section.

## 6.3 Reuters clusters and *dependency* clustering features

For the experiments in this section we use all the *dependency* features as described in 6.1.2 , generating a total of 1,673,744 features from the Reuters corpus for the lemmas to be clustered. Setting the feature frequency cut-off to 10 leaves us with a total of 339,473 features.

### 6.3.1 Development

Repeated runs of the K-means clustering algorithm with a batch size of 1000 and 'k-means++' as the method of initialization, gave only 20–30 clusters in the attempt of generating 100 clusters. We therefore increased the batch size to 5000, since that gave the specified number of 100 clusters. The following parameter values were used in the K-means algorithm: init = 'k-means++', n_init = 5, max_iter = 150 and batch_size = 10000.

Tables 6.8, 6.9 and 6.10 presents the development results on the WSJ 22, Weblogs and Emails sections, respectively.

**WSJ 22 with Reuters clusters from *dependency* features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 86.75 | 86.58 | 86.51 |
| *Form_simple* | **86.79** | 86.65 | 86.63 |
| *Form_all* | 86.73 | **86.79** | 86.75 |

Table 6.8: *Best LAS score for each combination of cluster number, MaltParser feature model and SVM parameter C on the WSJ 22 development section using Reuters clusters based on the dependency features. The baseline result was 86.72.*

**Weblogs with Reuters clusters from *dependency* features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 80.19 | 80.30 | 80.08 |
| *Form_simple* | 80.29 | 80.32 | 80.18 |
| *Form_all* | 80.31 | **80.34** | 80.27 |

Table 6.9: *Best LAS score for each combination of cluster number, MaltParser feature model and SVM parameter C on the Weblogs section using Reuters clusters based on the dependency features. The baseline result was 80.00.*

**Emails with Reuters clusters from *dependency* features**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 73.07 | 72.98 | 72.81 |
| *Form_simple* | 73.21 | 73.01 | 73.12 |
| *Form_all* | **73.22** | 73.11 | **73.22** |

Table 6.10: *Best LAS score for each combination of cluster number k, MaltParser feature model and SVM parameter C on the Emails section using Reuters clusters based on all the dependency features. The baseline result was 72.85.*

### 6.3.2 Held out testing

During development on the WSJ 22, two configurations of parameter values were optimal: the *Form_simple* model with k=10 and C= 0.0625, and the *Form_all* model with k=50 and C=0.0625. On the Weblogs section, the best configuration was equal to one of the two best configurations on WSJ 22, with the *Form_all* model, 50 clusters and a C-value at 0.0625. For the Emails section, again two configurations of parameter values were optimal: the *Form_all* model, 10 clusters, a C-value at 0.0625, and the *Form_all* model, 100 clusters and a C-value at 0.0625.

Considering that the *Form_all* model, 50 clusters and a C-value at 0.0625 performed best for both the WSJ 22 and Weblogs sections, we chose to use this during held-out testing on the WSJ 23, Answers, Newsgroups and Reviews sections. The held-out test results are presented in table 6.11.

| Held out testing with Reuters clusters | | | | |
|---|---|---|---|---|
| | WSJ Ont 23 | Answers | Newsgroups | Reviews |
| Baseline | 86.88 | 73.10 | 76.13 | 75.01 |
| Clusters | **87.16** | **73.58** | **76.97** | **75.43** |
| Difference | +0.28 | +0.48 | +0.84 | +0.42 |
| Significant | yes | yes | yes | yes |

Table 6.11: *Held-out testing LAS on WSJ 23 and the labeled SANCL Answers, Newsgroups and Reviews sections, using Reuters clusters based on all the dependency features.*

The WSJ 23 section had an improvement of 0.28 percentage points over the baseline model, a difference being statistically significant. The Answers section had an improvement of 0.48 percentage points, Newsgroups with 0.84 percentage points and the Reviews section with an improvement of 0.42 percentage points over the baseline model. All of these differences were also statistically significant.

With the use of clusters generated from the Reuters corpus, we compare the held-out testing results using the *path-to-root-pos* and *dependency* clustering features in table 6.12.

| Comparing held-out results using Reuters clusters | | | | |
|---|---|---|---|---|
| | WSJ Ont 23 | Answers | Newsgroups | Reviews |
| *path-to-root-pos* feature | 87.02 | 73.47 | 76.79 | 75.40 |
| Dependency features | **87.16** | **73.58** | **76.97** | **75.43** |
| Difference | 0.14 | 0.11 | 0.18 | 0.03 |

Table 6.12: *Comparing LAS on WSJ 23 and the labeled SANCL Answers, Newsgroups and Reviews sections, with Reuters clusters, based on the path-to-root-pos and dependency clustering features.*

We see that using the dependency clustering features gave a better parsing performance for all the held-out test sections. The smallest improvement is seen on the Reviews section, with a difference of 0.03 percentage points. None of the differences are, however, statistically significant.

From the parsing scores, the results clearly shows the difficulty in applying parsers to text outside the domain of the training data (WSJ sections 02-21), combined with the added noise expected to be found in web data text compared to newswire text. There is a clear drop in performance when parsing the labeled SANCL data compared to the WSJ data. While we see that cluster-informed parsers improves over the baseline across all data sets, we also see that the improvements are larger for the web data than for WSJ data. For Weblogs and Emails (development) the relative reductions of error rate (RER) is 1.7% and 0.96% respectively,

compared to 0.53% for WSJ section 22. During held-out testing, the gains of using cluster-informed parsers are even larger. From table 6.11 we see that when comparing the baseline and cluster-informed parser on Answer, Newsgroups and Reviews, we get error reductions of 1.78%, 3.52% and 1.68%, respectively. For WSJ 23, the RER is 2.13%.

## 6.4 In-domain clusters and the path-to-root-pos clustering feature

In the previous experiments, we generated clusters based on a vocabulary from the Reuters corpus and parsed both WSJ 23 and the web domains. Improvement was seen in parsing accuracy for the web data when comparing the baseline parser with the cluster-informed *Form_all* parser. Even greater gains in accuracy could be expected when using clusters generated from texts in the same domain that is to be parsed. In this and the next section we present the parsing experiments when running the K-means algorithm on the unlabeled SANCL data from their respective test domains. This means that, for example, the 4,060 sentences in the labeled Weblogs data is parsed using clusters generated for the 50,000 most frequent lemmas of the 524,834 sentences in the unlabeled Weblogs data.

During development, the labeled SANCL Weblogs and Email sections are parsed using cluster features based on unlabeled data from their corresponding SANCL domains. In the same vein, during held-out testing, the labeled SANCL Reviews, Answers and Newgroups sections are parsed based on clusters from their corresponding unlabeled SANCL domain data. When generating the cluster features, the feature frequency cut-off was set to 2. The vocabulary of lemmas for the unlabeled Answers section was 22,227 because of the smaller size of this data set, while the remaining four sections has a vocabulary size of 50,000. Table 6.13 presents the number of cluster features generated in total, and the actual number of features used for the lemmas to be clustered.

**Feature frequencies for the Unlabeled SANCL sections**

| | Weblogs | Emails | Answers | Newsgroups | Reviews |
|---|---|---|---|---|---|
| Total features | 720,934 | 790,062 | 53,760 | 975,304 | 1,340,942 |
| Used features | 251,765 | 322,379 | 11,548 | 280,385 | 301,853 |

Table 6.13: *Number of features generated from the unlabeled SANCL data sets using the path-to-root-pos clustering feature and a feature frequency cut-off at 2.*

### 6.4.1 Development

Repeated runs with 'k-means++' as the initialization with 100 clusters, gave 25 to 35% singleton clusters on both the Weblogs and Emails sections. We then chose 'random' as the initialization, since that gave a lower percentage of singleton clusters, ranging from 10 to 20 percent. In the attempt to

generate 200 clusters, the result was between 30 and 40% singleton clusters for both the Weblogs and Emails sections. This was the case with both 'random' and 'k-means++' as the method of initialization, with batch sizes at 1,000 and 10,000 for both methods. We will then only report parsing results based on 10, 50 and 100 clusters. The following parameter values were used in the K-means algorithm for generating clusters: init = 'random', n_init = 5, max_iter = 150 and batch_size = 1000.

The development results for the Weblogs and Emails sections are presented in tables 6.14 and 6.15, respectively.

**Weblogs with in-domain clusters from path-features**

| | 10 | 50 | 100 |
|---|---|---|---|
| Pos_simple | 80.18 | 80.03 | 80.03 |
| Form_simple | **80.31** | 80.08 | 80.11 |
| Form_all | 80.29 | 80.08 | 80.06 |

Table 6.14: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on the Weblogs development section using in-domain clusters based on the path-to-root-pos clustering feature. The baseline result was 80.00*

**Emails with in-domain clusters from path-features**

| | 10 | 50 | 100 |
|---|---|---|---|
| Pos_simple | 72.87 | 72.93 | 73.04 |
| Form_simple | 72.87 | 72.95 | 73.00 |
| Form_all | 72.94 | 72.95 | **73.15** |

Table 6.15: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on the Emails development section using in-domain clusters based on the path-to-root-pos clustering feature. The baseline result was 72.85*

### 6.4.2 Held out testing

The optimal configuration of parameters during development on the Weblogs section, was the *Form_simple* model with 10 clusters and a C-value at 0.0625. On the Emails section the parameter values were the *Form_all* model with 100 clusters and a C-value at 0.0625. In this case, we chose to perform held-out testing using both of these parameter configurations on the labeled SANCL Answers, Newsgroups and Reviews sections, since it is not possible to pick a single configuration in a principled way.

Table 6.16 presents the held-out testing results using the configuration of parameters that was best during development on the Weblogs section:

- Parsing model = *Form_simple*

- Number of clusters = 10
- c = 0.0625

**In-domain held-out testing using *Form_simple* with 10 clusters**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Clusters | **73.29** | **76.69** | **75.19** |
| Difference | +0.19 | +0.56 | +0.18 |
| Significant | yes | yes | yes |

Table 6.16: *Held-out testing LAS on the labeled SANCL Answers, Newsgroups and Reviews sections using in-domain clusters based on the path-to-root-pos clustering feature. The configuration of parameter values used is the Form_simple model, 10 clusters and a C-value at 0.0625, which was best during development on the Weblogs section.*

Table 6.17 presents the held-out parsing results using the configuration of parameters that was best during development on the Emails section: Parsing model = *Form_all* , Number of clusters = 100 c = 0.0625

**In-domain held-out testing using *Form_all* with 100 clusters**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Clusters | **73.32** | **76.64** | **75.53** |
| Difference | +0.22 | +0.51 | +0.52 |
| Significant | yes | yes | yes |

Table 6.17: *The table presents the held-out testing on the labeled SANCL Answers, Newsgroups and Reviews sections using in-domain clusters based on the path-to-root-pos clustering feature. The configuration of parameter values used is the Form_all model, 100 clusters and a C-value at 0.0625, which was best during development on the Emails section*

To make comparison more convenient, the held-out scores for the two configurations are repeated in table 6.18

| Held out testing with in-domain clusters | | | |
|---|---|---|---|
| | Answers | Newsgroups | Reviews |
| *Form_simple* , 10 cl. | 73.29 | **76.69** | 75.19 |
| *Form_all* , 100 cl. | **73.32** | 76.64 | **75.53** |
| Difference | 0.03 | 0.05 | 0.24 |

Table 6.18: *The table presents the held-out testing on the labeled SANCL Answers, Newsgroups and Reviews sections, with in-domain clusters based on all the path-to-root-pos clustering feature. Two configurations of parameters performed equally during development, and both were then applied during held-out testing.*

The differences in parsing scores between using the two configurations are small for the Answers and Newsgroups, being 0.03 and 0.05 percentage points, respectively. These are not statistically significant. The difference is, however, larger on the Reviews section, at 0.24 percentage points. This is statistically significant. Overall, the best parsing performance is the result of using the *Form_all* model with 100 clusters in this experiment.

### 6.4.3 Comparing Reuters and in-domain held-out results

To see the difference between parsing the labeled SANCL Answers, Newsgroups and Reviews sections annotated with cluster labels from Reuters and in-domain clusters using the *path-to-root-pos* clustering feature, we present the held-out test results in table 6.19. The in-domain results are based on using the configuration of parameters with *Form_all* model and 100 clusters, which resulted in the best parsing score during held-out testing.

| Comparing held-out results using path-to-root-pos feature | | | |
|---|---|---|---|
| | Answers | Newsgroups | Reviews |
| Reuters clusters | **73.47** | **76.79** | 75.40 |
| In-domain clusters | 73.32 | 76.64 | **75.53** |
| Difference | 0.15 | 0.15 | 0.13 |

Table 6.19: *Comparing held-out testing LAS on the labeled SANCL Answers, Newsgroups and Reviews sections, with in-domain and Reuters clusters, based on the path-to-root-pos clustering feature.*

The use of Reuters clusters gave the best parsing performance on the Answers and Newsgroups test sections, while the use of in-domain clusters gave the best parsing performance on the Reviews section. None of the differences are, however, statistically significant. Since the differences in parsing performance between using clusters from the Reuters corpus and the in-domain clusters are rather small, we do not conclude whether one is better than the other. The parser using in-domain web clusters performs better than the parser using Reuters clusters in two out of five domains;

Emails (development) and Reviews (held-out testing). These are also the two domains with the largest unlabeled data sets, as seen in table 5.1. At the same time, we see that the Reuters Corpus is vastly larger than any of the unlabeled SANCL corpora. During parsing we have also seen that the optimal configuration of parameter values is the *Form_all* model with 100 clusters and C-value of 0.0625.

## 6.5 In-domain clusters and *dependency* features

In this experiment, we use in-domain clusters based on the *dependency* clustering features for parsing the web data instead of the *path-to-root-pos* clustering feature used in the previous section. During development, the labeled SANCL Weblogs and Email data sets are parsed using cluster features based on their corresponding unlabeled SANCL domains. During held-out testing, the labeled SANCL Reviews, Answers and Newgroups sections are parsed based on clusters from their corresponding unlabeled SANCL domains, like in the previous section. When generating the clustering features from the *dependency* templates described in section 6.1.2, the threshold was set to 2, using only the features with a total frequency of 2 or more. The vocabulary for the unlabeled Answers section was 22,227, while the remaining four SANCL sections had a vocabulary size of 50,000. The frequency of features is presented in table 6.20

**Feature frequencies for the Unlabeled SANCL sections**

| | Weblogs | Emails | Answers | Newsgroups | Reviews |
|---|---|---|---|---|---|
| Total features | 236,358 | 267,903 | 42,564 | 427,036 | 370,980 |
| Used features | 170,121 | 196,488 | 27,101 | 286,523 | 240,727 |

Table 6.20: *Number of features generated from each of the five SANCL domains, using the dependency features.*

### 6.5.1 Development

In order to generate 100 clusters during development, the batch-size was set to 10,000 and the method of initialization to 'k-means++'. A smaller batch size resulted in generating a number of clusters less than the 100 specified. The following parameter values were used in the K-means algorithm: init = 'k-means++', n_init = 5, max_iter = 150 and batch_size = 10,000.

Tables 6.21 and 6.22 presents the development results on the Weblogs and Emails sections.

**Weblogs development with in-domain clusters**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 80.15 | 80.06 | 80.09 |
| *Form_simple* | 80.18 | 80.23 | **80.29** |
| *Form_all* | 80.18 | 80.13 | 80.20 |

Table 6.21: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on the Weblogs development section using in-domain clusters based on the dependency features. The baseline result was 80.00.*

**Emails development with in-domain clusters**

| | 10 | 50 | 100 |
|---|---|---|---|
| *PoS_simple* | 72.96 | 73.08 | 73.03 |
| *Form_simple* | 73.05 | 73.15 | 73.14 |
| *Form_all* | 73.11 | 73.08 | **73.35** |

Table 6.22: *Best LAS for each combination of cluster number, MaltParser feature model and SVM parameter C on the Emails development section using in-domain clusters based on the dependency features. The baseline result was 72.85.*

### 6.5.2 Held out testing

During development on the Weblogs and Emails sections, there were two configurations of parameter values giving equal optimal performance. Both configurations included 100 clusters and a C-value of 0.0625. The difference was the parsing model, with the *Form_simple* model from development on the Weblogs section, and the *Form_all* model from development on the Emails section. During held-out testing on the Answers, Newsgroups and Reviews sections, we test both of these configurations as seen form tables 6.21 and 6.22.

**Held out testing with *Form_simple* using in-domain clusters**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Clusters | **73.39** | **76.74** | **75.37** |
| Difference | 0.29 | 0.61 | 0.36 |
| Significant | yes | yes | yes |

Table 6.23: *Held-out testing LAS on the labeled SANCL Answers, Newsgroups and Reviews sections using the Form_simple model, 100 clusters, and a C-value at 0.0625, with in-domain clusters based on the dependency clustering features.*

**Held out testing with *Form_all* using in-domain clusters**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Clusters | **73.39** | **76.87** | **75.51** |
| Difference | 0.29 | 0.74 | 0.50 |
| Significant | yes | yes | yes |

Table 6.24: *Held-out testing LAS on the labeled SANCL Answers, Newsgroups and Reviews sections using the Form_all model, 100 clusters, and a C-value at 0.0625, with in-domain clusters based on the dependency clustering features.*

The held-out parsing results for the two configurations are presented in table 6.25

**Held out testing with in-domain clusters from *dependency* features**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| *Form_simple* , 100 cl. | **73.39** | 76.74 | 75.37 |
| *Form_all* , 100 cl. | **73.39** | **76.87** | **75.51** |

Table 6.25: *Held-out testing LAS on the labeled SANCL Answers, Newsgroups and Reviews sections, with in-domain clusters based on the dependency clustering features. Two configurations of parameters performed equally during development, and both were then applied during held-out testing.*

We see that the configuration of parameter values with the *Form_all* model gives the best parsing performance. This is in line with previous results; this model seems to consistently perform the best.

### 6.5.3 Comparing Reuters and in-domain results

We compare the held-out parsing results from using Reuters clusters and in-domain clusters when parsing the labeled SANCL Answers, Newsgroups and Reviews sections in table 6.26

**Comparing Reuters vs in-domain**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Reuters clusters | **73.58** | **76.97** | 75.43 |
| In-domain clusters | 73.39 | 76.87 | **75.51** |

Table 6.26: *Comparing held-out LAS on the labeled SANCL Answers, Newsgroups and Reviews sections, with in-domain and Reuters clusters, based on the dependency features.*

Using Reuters clusters gives the best parsing performance for the Answers and Newsgroups sections, while using in-domain clusters gives the best performance on the Reviews section. This is consistent with the results when using the *path-to-root-pos* clustering feature. In a later round of experiments, we therefore wanted to see whether we could compensate for this difference in corpora size by clustering all the unlabeled SANCL data combined, while still hoping to see positive effects of using data closer to the test domain.

## 6.6 Comparison between the path-to-root-pos and *dependency* clustering features

We have performed parsing experiments using clusters generated from the *path-to-root-pos* and *dependency* clustering features. They give a different distribution of lemmas among the clusters, and in particular, using the *path-to-root-pos* feature had a tendency of generating more singleton clusters than using the *dependency* clustering features. Table 6.27 presents the held-out testing results, performed with Reuters and in-domain clusters, applying the *path-to-root-pos* and dependency clustering feature functions.

**Comparing all held-out parsing results**

| | WSJ Ont 23 | Answers | Newsgroups | Reviews |
|---|---|---|---|---|
| Baseline | 86.88 | 73.10 | 76.13 | 75.01 |
| Reuters path root | 87.02 | 73.47 | 76.79 | 75.40 |
| Reuters *dependency* | **87.16** | **73.58** | **76.97** | 75.43 |
| In domain path root | N/A | 73.32 | 76.64 | **75.53** |
| In domain *dependency* | N/A | 73.39 | 76.87 | 75.51 |

Table 6.27: *Comparing held-out LAS using the path-to-root-pos and dependency clustering features, based on Reuters clusters and in-domain clusters.*

Parsing with cluster features generated from the *dependency* features, contributed to an overall better parsing performance when using Reuters clusters. Using in-domain clusters, the parsing performance was best when the clusters were generated by using the *dependency* features for the Answers and Newsgroups sections. The result was, however, marginally better with the *path-to-root-pos* feature for the Reviews section. Parsing the Reviews section based on in-domain clusters also gave better results than using Reuters clusters both for the *path-to-root-pos* and *dependency* clustering features.

Overall is seems like clustering lemmas generated from the Reuters corpus using the *dependency* features, contributed to an overall best parsing performance.

## 6.7 Clustering all the five unlabeled SANCL domains

The experiments using in-domain clusters were based on generating clusters from each of the five unlabeled SANCL domains separately. A motivation for the experiments in this section is to see whether using word clusters generated from all the five unlabeled SANCL sections together gives better parsing performance than using clusters from each domain individually. We generate clusters from all the five unlabeled SANCL domains concatenated into one data set, then perform parsing on each of the five labeled SANCL sections separately. Using the *dependency* clustering features, generated a total of 891,667 features. With the feature cut-off set to 3, there were 375,793 features left to be used for the 50,000 lemmas to be clustered. Using a feature cut-off at 3 for the *path-to-root-pos* clustering feature, 475,386 features were used out of a total of 3,070,477 features.

The configuration of parameters used, are the *Form_all* model, k = 100, and a C-value at 0.0625. This configuration of parameter values has been the overall optimal configuration during the previous development sessions.

For generating clusters with the *dependency* clustering features and *path-to-root-pos* clustering feature, we used the following parameter values for the K-means algorithm: init = 'k-means++', n_init = 5, max_iter = 150 and batch_size = 10,000.

The parsing results using the *path-to-root-pos* and *dependency* clustering features respectively, are presented in table 6.28.

**Held out testing with all SANCL domain clusters**

| | Weblogs | Emails | Answers | Newsgroups | Reviews |
|---|---|---|---|---|---|
| Baseline | 80.00 | 72.85 | 73.10 | 76.13 | 75.01 |
| Path-to-root-pos | 80.15 | 73.10 | 73.50 | **77.04** | **75.62** |
| Dependency | **80.26** | **73.27** | **73.52** | 76.94 | 75.53 |
| Difference | 0.09 | 0.17 | 0.02 | 0.10 | 0.09 |

Table 6.28: *Comparing LAS on the labeled SANCL Weblogs, Emails, Answers, Newsgroups and Reviews sections for the path-to-root-pos feature and dependency features. The clusters were generated from all the five SANCL domains concatenated.*

Using *dependency* features for the clustering contributes to a better parsing performance on the Weblogs, Emails and Answers sections, while the use of the *path-to-root-pos* clustering feature contributes to the best parsing performance on the Newsgroups and Reviews sections, but the differences are mostly small.

For easier comparison, table 6.29 presents the parsing results for each of the five labeled SANCL sections, using both in-domain clusters and clusters based on all the five sections concatenated. All the parsing scores are based on using the *Form_all* parsing model, 100 clusters and a C-value at 0.0625.

| Comparing all held-out parsing results | | | | | |
|---|---|---|---|---|---|
| | Weblogs | Emails | Answers | Newsgroups | Reviews |
| Baseline | 80.00 | 72.85 | 73.10 | 76.13 | 75.01 |
| In domain path | 80.06 | 73.15 | 73.32 | 76.64 | 75.53 |
| All SANCL path | 80.15 | 73.10 | 73.50 | **77.04** | **75.62** |
| In domain dep | 80.20 | **73.35** | 73.39 | 76.87 | 75.51 |
| All SANCL dep | **80.26** | 73.27 | **73.52** | 76.94 | 75.53 |

Table 6.29: *Comparing LAS using the path-to-root-pos and dependency clustering features, based on in-domain clusters and clusters from all the five SANCL domains concatenated.*

From table 6.29 we see that using clusters based on all the five SANCL sections leads to a better parsing score than using clusters based on each section individually when clustering based the *dependency* features. The exception is the Emails section, where in-domain clusters yields to the best parsing score, both when using the *path-to-root-pos* and *dependency* clustering features. The difference is, however, marginally at 0.05 percentage points for the *path-to-root-pos* feature and 0.08 points for the *dependency* features.

Using clusters based on all the SANCL sections generated with *dependency* features contributes to a better parsing score for the Weblogs, Emails and Answers sections, compared to using the *path-to-root-pos* feature. The parsing score is, however, better using the *path-to-root-pos* clustering feature when generating clusters for parsing the Newsgroups and Reviews sections. Using in-domain clusters based on *dependency* features contributed to a better parsing score than using the *path-to-root-pos* feature for all the sections, except the Reviews section, where generating clusters with the *path-to-root-pos* feature contributed to a marginally better parsing score, 0.02 percentage points higher.

When testing for statistical significance on the three held-out domains (Answers, Newsgroups and Reviews), none of the differences between the Reuters and all SANCL experiment sessions with *dependency* features are detected as being statistically significant. In the Reuters experiments (table 6.11), we see that the parsing scores were 73.58% (Answers), 76.97% (Newsgroups) and 75.43% (Reviews). The use of all-in-one SANCL clusters gave correspondingly 73.52%, 76.94% and 75.53%.

For the *dependency*-based word clusters, it is not possible to conclude anything about which data set provides the optimal source for generating these clusters for the parser. However, it is clear that whichever data set is used, the cluster informed *Form_all* parser improves significantly over the baseline parser.

We have in the previous sections experimented with vocabularies consisting of 50,000 lemmas. In order to study the effect of using a reduced vocabulary, we repeat the experiments using the *dependency* features, only this time, clustering only 5,000 and 10,000 lemmas instead. The next section will present the results on comparing the use of 50,000 vs 5,000 and 10,000

lemmas. We will also report results for a round of experiments using agglomerative clustering.

## 6.8 Testing with smaller vocabularies and agglomerative clustering

In order to assess the effect of vocabulary size, we generated vocabularies of 10,000 lemmas contrary to using 50,000 as in the previous experiments. This was done for the Reuters corpus and all the five unlabeled SANCL domains, including the five domains concatenated into data set. We repeated this only using the *dependency* clustering features. This section will then be a repetition of the experiments using the *dependency* features, now only with clustering vocabularies with 10,000 lemmas instead. We then compare the results with using a 50,000 vocabulary. Note that the Answers vocabulary only consisted of 22,227 lemmas, because of the smaller size of this data set. We will also try using an even smaller vocabulary with the 5,000 most frequent lemmas from the Reuters corpus to see how that compare.

For generating feature-matrices for the lemmas to be clustered, we used the same feature cut-offs as in the experiments with 50,000 vocabularies. We also used the same configuration of parameters: the *Form_all* parsing model, a C-value at 0.0625 and 100 clusters. The K-means parameter values were the same as for clustering 50,000 lemmas, except for the batch-size being reduced to 5000.

In section 6.1.5, we performed 10 repeated runs on the K-means algorithm with the same parameter values as input and parsed WSJ section 22 to see how this affected parsing performance using the various clustering results. This is because K-means is a non-deterministic algorithm, due to the random initialization of the seeds. We repeated this with the vocabulary of 10,000 lemmas, and compared the results with runs using a vocabulary of 50,000 lemmas. The comparison of these runs are shown in table 6.30.

**Basic statistics for ten repeated clustering and parsing runs**

| | 10k vocabulary | 50k vocabulary |
|---|---|---|
| Min | 86.65 | 86.64 |
| Max | 86.85 | 86.89 |
| Mean | 86.78 | 86.78 |
| Median | 86.81 | 86.81 |
| Standard Deviation | 0.05903 | 0.09476 |
| Variance | 0.003484 | 0.008979 |

Table 6.30: *The basic statistics of LAS after running K-means ten times with the same vocabulary and parameters, comparing clustering vocabularies of 10,000 and 50,000 lemmas.*

From table 6.30 we see that the average parsing score is the same after parsing WSJ 22 based on clusters from ten different K-means clusterings. There is, however, less variance in the runs based on the vocabulary of 10,000 lemmas. Using a smaller vocabulary is likely to cause less variation in the clustering, leading to less variation in the parsing accuracies when parsing with a cluster-based model. The following presents the comparison on how the use of different vocabulary sizes for clustering affects parsing performance on the remaining data sets.

We did not repeat the same extensive development session as with the 50,000 vocabulary for finding the optimal configuration of parse model, C-value and number of clusters. Instead, we tested C-values using only the *Form_all* model with 10 and 50 clusters. Given the clusters generated based on the dependency features for the 10,000 most frequent lemmas from the Reuters corpus, we started on the WSJ 22 section. The best combination was 100 clusters and a C-value at 0.0625. Development on the Weblogs section revealed the same values. With in-domain clusters, also using only the *Form_all* model, with 50 and 100 clusters during development, the best combination was 50 clusters and C=0.0625 for Weblogs. For Emails, the best combination was 100 clusters and C=0.0625. We then used the same configuration as seen before with the *Form_all* parse model, a C-value at 0.0625 and 100 clusters during held-out testing.

**Held out testing with Reuters clusters**

| | WSJ Ont 23 | Answers | Newsgroups | Reviews |
|---|---|---|---|---|
| Baseline | 86.88 | 73.10 | 76.13 | 75.01 |
| Clusters 10k voc | 87.15 | 73.37 | 76.55 | 75.12 |
| Clusters 50k voc | **87.16** | **73.58** | **76.97** | **75.43** |
| Difference | +0.01 | +0.21 | +0.42 | +0.31 |
| Significant | no | yes | yes | yes |

Table 6.31: *Comparing held-out LAS based on clusters from the dependency features, using vocabularies with 10,000 and 50,000 lemmas from Reuters.*

When using Reuters data, table 6.31 shows that using a vocabulary of 50,000 lemmas yields better performance than when using only 10,000. The difference is, however, marginal for WSJ 23, but significant for the three labeled SANCL domains.

We also repeat the experiments using in-domain clusters, where we generate vocabularies of 10,000 lemmas from each of the unlabeled SANCL domains, and perform parsing on the corresponding labeled SANCL domains.

**Held out testing with in-domain clusters from *dependency* features**

| | Answers | Newsgroups | Reviews |
|---|---|---|---|
| Baseline | 73.10 | 76.13 | 75.01 |
| Clusters 10k voc | **73.49** | 76.55 | 75.32 |
| Clusters 50k voc | 73.39 | **76.87** | **75.51** |
| Difference | -0.10 | +0.32 | 0.19 |
| Significant | no | yes | no |

Table 6.32: *Comparing held-out LAS based on clusters from the dependency features, using vocabularies with 10,000 and 50,000 lemmas from Reuters.*

Table 6.32 compares parsing performance when using vocabularies of 50k and 10k extracted from the various SANCL domains. For Newsgroups and Reviews we see that more is better, while the smaller vocabulary works better for Answers.

**Held out testing with all SANCL clusters from *dependency* features**

| | Weblogs | Emails | Answers | Newsgroups | Reviews |
|---|---|---|---|---|---|
| Baseline | 80.00 | 72.85 | 73.10 | 76.13 | 75.01 |
| 10k voc | 80.20 | 73.05 | 73.40 | 76.66 | 75.32 |
| 50k voc | **80.26** | **73.27** | **73.52** | **76.94** | **75.53** |
| Difference | +0.06 | +0.22 | +0.12 | +0.28 | +0.21 |
| Significant | no | yes | no | yes | yes |

Table 6.33: *Comparing held-out LAS based on clusters from from the dependency features, using vocabularies with 10,000 and 50,000 lemmas from the concatenated SANCL sections.*

When using the concatenated unlabeled SANCL data, however, table 6.33 shows that the larger vocabulary yields the best results across all domains.

## 6.9 Agglomerative clustering

We have in the previous sections been experimenting with the K-means clustering algorithm. As an alternative we try the hierarchical agglomerative clustering functionality provided in SciPy. In order to make these experiments computationally feasible, we use here an even smaller vocabulary, consisting of only the 5,000 most frequent lemmas from the Reuters corpus, clustered based on the *dependency* features. The reason is SciPy's use of dense feature vectors where a value for each possible feature for the lemmas is recorded, and this will not scale well because of the high number of features. Given a specific lemma, a feature count will be stored for every possible feature. For features not used by the lemma, a zero is then stored. Since most features are not used, the vector will then consist mainly of zero

values, and then wasting space. This dense representation is required by the agglomerative algorithm in SciPy for calculating the distances between the vectors. For improving scalability, we calculate the distance matrix input to the agglomerative clustering algorithm in SciPy ourselves, providing a *condensed distance matrix* as input. For each pair of vectors *x* and *y*, we calculate the Euclidean distance only once for each pair since *dist (x,y) = dist (y,x)*. If this had been done in the naive way, calculating both *dist (x,y)* and *dist (y,x)*, the result would be a *n* x *n* matrix given *n* vectors. In this matrix we are interested only in the lower left part (or upper right part) mirrored by the diagonal.

Since the distance measure is symmetrical, the value in cell[i,j] = cell[j,i]. In general, for storing values in a *n* x *n* matrix, a one-dimensional array of length n *((n-1)/2) could be used. We will then use this condensed distance matrix for storing vector distances as input o the hierarchical clustering algorithm, instead of the naive *squareform* matrix. Even so, computing all of these pairwise similarities is a costly operation. The linkage criterion used is group-average hierarchical clustering. The vectors are also length normalized.

We perform held-out testing on the labeled SANCL sections, in addition to WSJ 23 for both clusters based on both agglomerative and K-means clustering. As before, we apply the configuration with the *Form_all* model, a C-value at 0.0625 and 100 clusters for K-means. The agglomerative clustering contained as much as 52 singleton clusters out of 100 in total. The distribution of lemmas across the clusters were then much less evenly distributed than in the K-means clusters. K-means generated no singleton clusters.

**Held out testing with Reuters clusters**

| | WSJ Ont 23 | Answers | Newsgroups | Reviews |
|---|---|---|---|---|
| Baseline | 86.88 | 73.10 | 76.13 | 75.01 |
| K-means 5k voc | **87.02** | **73.38** | **76.54** | **75.13** |
| Agglomerative 5k voc | 86.84 | 73.29 | 76.28 | 74.94 |

Table 6.34: *Comparing held-out LAS based on clusters from the dependency features, using K-means and agglomerative clustering on a vocabulary with 5,000 lemmas from Reuters.*

From table 6.34 we see that the clusters generated from the hierarchical agglomerative clustering yields lower accuracies across all domains compared to the the use of K-means clusters.

In the next section, we investigate the use of the *dependency*-based clusters compared to *n*-gram Brown clusters (explained in section 3.2.2) as used in most previous studies explained in chapter 4.

## 6.10 Comparing results with Øvrelid and Skjærholt

In this section, we compare the results obtained using our *dependency*-based clusters and the Brown clusters used by (Øvrelid and Skjærholt, 2012) and several other previous studies. We report the results of parsing web data from the OntoNotes corpus, release 4.0, more specifically the WSJ 23 test section, the *Eng* section, containing general English web data (71,500 tokens) and the *Sel* section (279,000 tokens), containing selected sentences for improving sense coverage in the corpus. (Øvrelid and Skjærholt, 2012). Vocabularies of the 50,00 most frequent lemmas from both the Reuters corpus and all the five SANCL domains concatenated are used, then clusters are generated using the *dependency* features, applying the K-means algorithm with init = 'k-means++', n_init = 5, max_iter =150 and a batch_size of 5,000.

As to isolate the effect of the clustering approach as best as possible, we here use the same version of MaltParser (v.1.4.1), as used by Øvrelid and Skjærholt (2012). The configuration of parameters we use are the optimal values found during development in our previous experiments, which is the *Form_all* model, 100 clusters and C-value at 0.0625, and apply models based on both the Reuters clusters and the all-in-one SANCL clusters.

The parsing results presented by Øvrelid and Skjærholt (2012) are to be considered development results, since they are based on the best parsing score after tuning the model parameters directly on the data. The parameters include the feature model and number of clusters. Our parsing scores presented in this section, are based on the aforementioned optimal parameter values found during development in our previous experiments, and then to be considered held-out testing. Table 6.35 present the parsing results using data with gold-standard part-of-speech tags.

**Comparing results using gold tags**

| | WSJ Ont 23 | Eng | Sel |
|---|---|---|---|
| Baseline Øvrelid (2012) | 89.27 | 83.89 | 83.61 |
| Baseline | 89.20 | 84.05 | 83.74 |
| Brown Øvrelid (2012) | 89.05 | 83.36 | 83.09 |
| Reuters, dep feats | **89.41** | **83.96** | 83.56 |
| All Sancl, dep feats | 89.13 | **83.96** | **83.79** |

Table 6.35: *Comparing LAS with Øvrelid and Skjærholt (2012), using data sets with gold standard part-of-speech tags.*

The baseline parser, trained with a C-value of 0.1 gave the same results as Øvrelid and Skjærholt (2012) when parsing the WSJ 23 and sel data. For the category *Eng*, the result was slightly higher at 83.92. We then retrained a baseline parser, this time using a C-value of 0.0625. This resulted in a small decrease of 0.07 percentage points in the parsing score for WSJ 23, and an improvement of 0.16 points for the category *Eng*. For the category *Sel*, there was an improvement of 0.13 percentage points.

Based on clusters from the Reuters corpus and parsing using the *Form_all* model, we achieved improvements of 0.36 percentage points for the WSJ 23 section, 0.60 points for the category *Eng* and 0.47 points for the category *Sel*, when comparing our results with (Øvrelid and Skjærholt, 2012). Compared to our baseline results, there was an improvement of 0.21 percentage points for the WSJ 23 section. For the *Eng* and *Sel* categories, there was, however, a decrease in parsing scores in 0.09 and 0.18 percentage points for the *Eng* and *Sel* categories, respectively.

Using clusters based on a vocabulary of all the five unlabeled SANCL domains concatenated, gave a lower parsing score for the WSJ 23 data compared to using cluster from the Reuters corpus. The score was, however, equal for the category *Eng* and improved with 0.23 percentage points for the category *Sel*. Compared to our baseline parser, we did not see any improvements for the WSJ 23 and *Eng* category, although a small increase of 0.05 percentage points for the category *Sel*.

Overall, for the *Eng* and *Sel* categories, clustering lemmas from the SANCL domains contributed to a better parsing performance, compared to using clusters of lemams from the Reuters corpus.

In addition to the parsing experiment using data with gold standard part-of-speech tags, we the data were part-of-speech tagged using SVMTool (version 1.3.1) with the pre-trained model for English based on the Wall Street Journal corpus (English WSJ) with the following settings:

- Tagging direction -S = LRL (Left Right Left)
- Strategy -T = 4

When parsing with a trained baseline parser with a C-value at 0.1, the default value used in (Øvrelid and Skjærholt, 2012). We did not in this case achieve the same parsing scores as reported in (Øvrelid and Skjærholt, 2012), despite using the same version and parameters when running SVMTool for part-of-speech-tagging. The exact reason for this remains unknown, but we expect there might be some inaccuracies in the WSJ data used in the experiment of (Øvrelid and Skjærholt, 2012). The difference in parsing score between the baseline and cluster based models are somewhat large.

**Comparing results using automatic tags**

| | WSJ Ont 23 | Eng | Sel |
|---|---|---|---|
| Baseline Øvrelid (2012) | 86.24 | 76.99 | 74.84 |
| Baseline | 86.67 | 78.45 | 76.02 |
| Brown Øvrelid (2012) | 86.67 | 78.30 | 75.82 |
| Reuters, dep feats | **86.98** | 78.71 | 76.23 |
| All Sancl, dep feats | 86.90 | **78.79** | **76.30** |

Table 6.36: *Comparing LAS with Øvrelid and Skjærholt (2012), using data sets with automatic part-of-speech tags generated by SVM-Tool. In these experiments we trained parsers using the C-valuesthat was optimal during development.*

From table 6.36 we see that our parsing model based on lemmas from the Reuters corpus using the *dependency* clustering features performed best when parsing WSJ 23. The difference from the Brown-based model of (Øvrelid and Skjærholt, 2012) was an increase in parsing score of 0.25 percentage points. For the *Eng* and Sel categories, we achieved best results using lemmas from the five SANCL domains concatenated. Compared to (Øvrelid and Skjærholt, 2012) there was an increase of 0.49 percentage points for the *Eng* category, and 0.48 percentage points for the *Sel* category. Comparing our baseline models to the Brown models, we see that our baseline model performed equally as the Brown model for WSJ 23, and even better for the *Eng* and *Sel* categories. There is likely that some other factors not taken in consideration have affected the results from (Øvrelid and Skjærholt, 2012).

## 6.11 Cluster examples

With a vocabulary consisting of the 10,000 most frequent lemmas from the five unlabeled SANCL domains concatenated, we generated 100 clusters using the *dependency* features. For some selected clusters, we show in figures 6.5 and 6.6, the 12 lemmas in each cluster being closest to its centroid. Each lemma is also associated with its Euclidean distance to the centroid.

## 6.12 Summary

This chapter has given a general explanation of the experimental process and a description of the actual experiments. This includes tuning the parameters during development, a description of the cluster-based features, how the vocabulary of lemmas were constructed, construction of the feature matrix. We also explained the process of generating clusters, as well as performing paring and evaluation of the parsing results.

The second part of the chapter gave a description of each experiment performed. We generated cluster from the Reuters corpus, using both the *path-to-root-pos* and *dependency* clustering features and performed parsing on the WSJ data as well as web data, more specifically the labeled SANCL domains. We also performed in-domain experiments, by generating clusters from the same domain that is to be parsed, more specifically generating clusters from the unlabeled SANCL domains and parsing each respective labeled SANCL domain. We tested the use of smaller vocabularies with 5,000 and 10,000 lemmas, respectively and reported significant differences compared to using a vocabulary of 50,000 lemmas. The use of agglomerative clustering contributed to a lower parsing performance compared to using K-means clusters. A comparison was also made to the results from (Øvrelid and Skjærholt, 2012) who used Brown clusters in their experiments.

**SANCL domains clusters**

```
3: reveal(0.14), condemn(0.15), deny(0.15), weigh(0.16)
   conclude(0.16), discover(0.17), reject(0.17),
   dismiss(0.17), begin(0.17), predict(0.17),
   demonstrate(0.17), show(0.18),

8: employ(0.15), organize(0.16), disable(0.16),
   involve(0.17), qualify(0.17), furnish(0.17),
   motivate(0.18), prescribe(0.19), bear(0.19),
   renovate(0.19), elect(0.20), certify(0.20),

10: weak(0.07), dangerous(0.09), careless(0.09),
    stupid(0.10), awkward(0.10), stale(0.10),
    scary(0.11), lazy(0.11), messy(0.11),
    inept(0.11), painful(0.11), sloppy(0.11),

11: ein(0.36), aber(0.36), dengan(0.41), lebih(0.44),
    dalam(0.46), bei(0.48), untuk(0.49), bagi(0.50),
    auf(0.53), +90(0.55), seiten(0.56),
    benutzerhandbuch(0.60)

26: brian(0.05), anna(0.05), laura(0.05), frank(0.06),
    joe(0.06), heather(0.06), lisa(0.06), barbara(0.06)
    donna(0.06), doug(0.06), keith(0.06), steve(0.06)

30: association(0.12), committee(0.12),
    foundation(0.13), festival(0.14), centre(0.14),
    tavern(0.15), center(0.15), republic(0.16),
    kingdom(0.17), empire(0.17), basin(0.18),
    federation(0.18),

31: kinda(0.19), disgusting(0.21), sooooo(0.21),
    soo(0.22), darn(0.22), eh(0.23), dont(0.23),
    everytime(0.23), soooo(0.23), lol(0.24),
    yea(0.24), cant(0.24),

36: boston(0.07), napa(0.08), riverside(0.08),
    hollywood(0.08), detroit(0.08), maine(0.08),
    lancaster(0.08), sd(0.09), phoenix(0.09),
    monroe(0.09), burlington(0.09), oc(0.09),
```

Figure 6.5: *Example of clusters of lemmas from the All SANCL vocabulary generated using the dependency features. The 12 lemmas in each cluster being closest to its centroid are shown, associated with their Euclidean distance.*

**SANCL domains clusters**

```
47: greatest(0.09), newest(0.11), longest(0.11),
    oldest(0.12), smallest(0.12), latest(0.12),
    strongest(0.13), biggest(0.14), finest(0.15),
    lowest(0.15), highest(0.15), friendliest(0.16),

51: miller(0.06), wilson(0.07), johnson(0.07),
    williams(0.09), anderson(0.09), carter(0.09),
    thompson(0.10), turner(0.10), bowen(0.10),
    harris(0.11), taylor(0.11), davies(0.11),

55: salesperson(0.13), hostess(0.13), salesman(0.15),
    pharmacist(0.15), receptionist(0.15),
    repairman(0.15), brisket(0.15), installer(0.15),
    owner(0.16), manger(0.16), crust(0.17), steak(0.18)

73: cafe(0.10), bank(0.10), gallery(0.10), castle(0.11)
    dam(0.11), club(0.11), bridge(0.12), grill(0.12),
    church(0.12), park(0.12), library(0.12),
    index(0.13),

79: layer(0.12), element(0.14), member(0.14),
    component(0.14), rule(0.15), variation(0.16),
    consequence(0.17), issue(0.17), dimension(0.17),
    prospect(0.17), result(0.18), provision(0.18),

86: a.(0.15), d.(0.15), p.(0.16), f.(0.16), c.(0.17),
    lt(0.18), h.(0.18), piper(0.21), n.(0.21),
    walter(0.22), bg(0.22), xxx(0.22),

93: 02:00:00(0.12), 08:00:00(0.14), 09:00:00(0.14),
    01:00:00(0.14), 03:00:00(0.14), 07:00:00(0.15),
    02:30:00(0.15), 01:30:00(0.15), 04:00:00(0.16),
    05:00:00(0.16), 04:30:00(0.17), 06:00:00(0.17),

95: june(0.10), march(0.10), sept(0.10), february(0.11)
    december(0.11), july(0.12), september(0.12),
    jan.(0.12), august(0.12), january(0.12), dec.(0.13)
    april(0.13),
```

Figure 6.6: *Example of clusters of lemmas from the All SANCL vocabulary generated using the dependency features. The 12 lemmas in each cluster being closest to its centroid are shown, associated with their Euclidean distance.*

# Chapter 7

# Conclusion

## 7.1 Conclusion and further work

The aim of this thesis has been to study the effect of using a semi-supervised approach with cluster-based features in MaltParser, a data-driven parser for dependency parsing, to see how the addition of these features contributed to parsing performance compared to only using a set of baseline features. We first parsed large corpora of unlabeled text, converting it into a dependency representation. This was then used as a basis for extracting clustering features used as input to the K-means algorithm, clustering lemmatized word forms. By re-training a cluster-informed parser on labeled data, this time using features from the clusters generated, parsing accuracy was shown to improve significantly over a baseline parser on both in-domain and out-of-domain tests, including web text from different domains. Our use of syntactically informed clusters also compare favorably to previous studies using the *n*-gram based Brown clusters.

We have previously mentioned that a main factor in dependency parsing is the assignment of word-to-word relations. Statistics of relations between word forms in the training data provides important information. Since these statistics regarding the relations are sparse, strategies for generalizing over word forms have been investigated in several previous studies. Chapter 4 reviewed some previous work on using word clusters in the attempt of improving parsing accuracy on statistical parsers, using clusters generated by the ngram-based Brown hierarchical clustering algorithm (Brown et al., 1992). Among these are the study reported by Koo et al. (2008), who performed semi-supervised dependency parsing experiments in English and Czech by using cluster-based features in parsers, trained using the averaged perceptron (Koo et al., 2008). For statistical constituent parsing in French, Candito and Seddah (2010) also used clusters generated by the Brown algorithm. These clusters consisted of lemmas and part-of-speech tagged lemmas (Candito and Seddah, 2010). Øvrelid and Skjærholt (2012) used Brown clusters in a series of web dependency parsing experiments, using MaltParser. The aim was to study the benefit of clustering from the perspective of domain-adaption and to

improve dependency parsing of various English web data. We maintain a similar focus in this thesis. In section 6.10, we performed a comparison experiment, reporting the results of parsing some of these data. The purpose was to compare results obtained using our clusters and the Brown clusters. Instead of using Brown clusters, Sagae and Gordon (2009) used parsed data for creating clusters based on syntactic similarity. Their goal was to improve the accuracy of a fast dependency parser by using a corpus previously annotated with a slower constituent parser (Sagae and Gordon, 2009). Our experiments also aimed at improving dependency parsing with syntactically informed clusters using parsed data, but using only one single parser, thereby introducing an element of self-training.

We performed experiments using several different data sets, described in more detail in chapter 5, both for generating word clusters and for performing parsing. The SANCL 2012 shared task on parsing English web data (Petrov and McDonald, 2012), provided both labeled and unlabeled data from five different domains in the English Web Treebank (Bies et al., 2012), in addition to the WSJ (OntoNotes version). For parsing, we used the labeled data, and for generating clusters of words, we used the unlabeled data, reporting both development and held-out results. In addition, we also used the Reuters corpus volume 1 (Rose et al., 2002) for as a source for word clustering.

In an initial experiment, we investigated the effect of using gold-standard vs automatically assigned part-of-speech tags. We trained two baseline versions of the parser using WSJ section 02-21, first using the standard gold tags provided with the data, then a second time replacing the gold tags with automatically generated tags by SVMTool (Giménez and Màrquez, 2004). In both runs, we then parsed WSJ sections 22 and 23 using automatically generated tags from SVMTool. The results showed a great advantage in training with automatically assigned tags. For all our further experiments, we then used automatically assigned part-of-speech tags in the data sets both for training and testing.

During parsing with MaltParser, we tested different parsing feature models for assessing the effect of using cluster-informed parsers compared to a baseline parser. The baseline model described by Foster et al. (2011) contains the baseline set of features used by MaltParser. We also tested three extended feature models, described by Øvrelid and Skjærholt (2012). These were the *PoS_simple*, *Form_simple* and *Form_all* models, all having the features from the baseline model, in addition, to being expanded with features for retrieving the cluster label for the data to be parsed. Before final held-out testing in the various experiments, we performed development sessions for tuning various model parameters, parsing development data with the three extended feature models. The development data sets used, were the Weblogs and Emails sections from the EWT, as well as WSJ section 22. As the preliminary experiments on the development data showed, the *Form_all* feature model was consistent in outperforming the *PoS_simple* and *Form_simple* models. The *Form_simple* model also performed better than the *PoS_simple* model. As explained in chapter 2, the *Form_all* model has all the features from the *Form_simple* model. It is also the only model of

the three extended models that uses conjunctive features by merging into a new feature information from two different tokens. One can then argue that these merge-based features seems fruitful for achieving additional parsing performance.

Another important factor in terms of development we investigated was model tuning. As mentioned in chapter 6, the C-value is used by the support vector machine (SVM). It works as a penalty parameter, allowing the decision margin to make classification mistakes. This parameter has been empirically tuned for all parser configurations on the development data. For each of the aforementioned development sets, we performed parsing using the four different parsing feature models. For each feature model, parsing was performed after training the parser with different C-values. With the three cluster-based parsing models, we also parsed the data annotated with cluster labels from both 10, 50 and 100 clusters. We tested parsers with clusters generated using two different sets of clustering features: the so-called *path-to-root-pos* features and the *dependency* features. When parsing with the baseline model, the C-value varied slightly across the different development sets. Overall, a C-value at 0.0625 was the one performing best with 100 clusters and *dependency* features, and then also applied when training parsers for subsequent held-out testing.

Several clustering algorithms are available, we have described the K-means algorithm, as well as hierarchical agglomerative clustering. An example of the latter, is the Brown algorithm, used for generating clusters in several of the previous works we have reviewed. Attempts with the hierarchical clustering algorithm from SciPy (Jones et al., 2001) implemented in Python, had some drawbacks. The lack of support for sparse matrices required dense feature vector representations of the lemmas to be clustered. That means storing a value for all possible features, including a zero for features not used by the lemma. In our experiments, the number of clustering features was typically around 300,000. These non-active features are highly frequent, and a lot of space is then wasted. Constraints on the number of lemmas to be clustered, as well as how many features they are represented with, motivated the use of a different clustering algorithm. As argued by Manning and Schütze (1999), non-hierarchical clustering is preferred for large data sets. The K-means algorithm is a non-hierarchical algorithm, being simple and leads often to sufficient results (Manning and Schütze, 1999). A Python implementation in scikit-learn (Pedregosa et al., 2011) has support for sparse matrices. Initially we tested the standard implementation of K-means, but this did not scale well in terms of time and memory usage for our 50,000 vocabulary to be clustered. A different implementation, the so-called mini-batch variant of K-means, performed much faster. Compared to the standard K-means who used at least 24 hours, it took only one hour. It was then the choice of algorithm during our experiments. An important factor with K-means is, however, the non-deterministic behavior. Several runs with the same input gives a variation of the outcome, as explained in section 6.1.5. We also report experimental results quantifying this effect, showing that it is noticeable though not substantial effect for our data sets. The hierarchical

clustering approach is, on the other hand, deterministic.

For generating clusters using the K-means algorithm, the number of clusters that is to be generated, needs to be specified as input to the algorithm. Words, or more specifically lemmatized word forms, were the objects to be clustered in our case. As mentioned in chapter 3, one can best assess the quality of a clustering extrinsically. In our case, that would be how the clustering affects parsing performance.

In our experiments, using the *dependency* clustering features gave clusters contributing to better parsing performance for the most part, compared to using the*path-to-root-pos* clustering feature. The differences were, however, not significant. This was the case using clusters from both the Reuters and SANCL data. We also noticed significant improvements in parsing performance from using a 50,000 vocabulary of lemmas compared to a 10,000 vocabulary. But importantly, regardless of the feature model used for clustering, the re-trained parser was shown to consistently outperform the baseline parser.

### 7.1.1 Further work

An approach for future work would be to perform *outlier detection*, prior to the clustering. Noisy objects may cause a lower quality of the clusters, and in our case cause provide a lower parsing accuracy. If an object regarded as an outlier is chosen to be an initial seed in the K-means algorithm, a singleton cluster will be formed since no other object will be assigned to this outlying observation Manning et al. (2008).

Overall, as mentioned above, our parsing experiments has shown that using the *dependency* clustering features gives slightly higher scores. Since the differences are not significant between using the two clustering features, it is, however, difficult to conclude whether one is better than the other. We presented in chapter 6 the two families of clustering feature types *path-to-root-pos* and the *dependency* features used for generating clusters for the purpose of this thesis. One could for future work expand the *path-to-root-pos* to generate paths from the target word (that is to be clustered), to all the words in the same sentence, not only the root word. Paths may or may not include the part-of-speech tag for the word at the end of the path.

A task for future projects is experimenting with the Maltparser feature models. More specifically, this could be experimentation with additional cluster-based features in the feature models used by the parser. One could for instance experiment with the *Form_all* model and expand it with more features for merging cluster labels between to tokens, or merging the cluster label from one token combined with the part-of-speech tag or word form from another token

Another avenue for follow-up work is to find a trade-off between the vocabulary size and the number of clusters. A smaller vocabulary would allow a higher number of clusters. We have in this experiment been using a rather large vocabulary of 50,000 lemmas. In our experiments, we created vocabularies based on words with different part-of-speech tags, but only using verbs, nouns and adjectives. A vocabulary would then

consist of words with any of these part-of-speech tags. When performing clustering in future work, one can choose to cluster lemmas based only on their part-of-speech. For instance, one can build a vocabulary consisting of only nouns or verbs separately. For the problem of scalability, this would be helpful. Given limited computational resources, the vocabulary size, number of features for the objects to be clustered and the number of clusters will be restricted. But when generating several separate K-means clusterings for different classes of part-of-speech, one can increase the number of clusters, allowing for a greater lexical coverage in the clusters.

# Bibliography


Anne Abeillé, Lionel Clément, and François Toussenel. *Treebanks: Building and Using Parsed Corpora*, chapter Building a Treebank for French. Kluwer, Dordrecht, 2003.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. English Web Treebank LDC2012T13, 2012.

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

Leonard Bloomfield. *Language*. Holt, New York, 1933.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December 1992.

Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *In Proc. of CoNLL*, pages 149–164, 2006.

Marie Candito and Djame Seddah. Parsing word clusters. In *Proceedings of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 76–84, Los Angeles, CA, 2010.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139, Seattle, WA, 2000.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. Brown laboratory for linguistic information processing (BLLIP) 1987–89 WSJ corpus release 1 LDC2000T43, 2000.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Association for Computational Linguistics -02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Jennifer Foster, Özlem Çetinoğlu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. #hardtoparse: POS tagging and parsing the twitterverse. In *Proceedings*

*of the AAAI Workshop on Analysing Microtext*, pages 20–25, San Francisco, CA, 2011.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Machine Learning*, pages 277–296, 1998.

Victoria Fromkin, Robert Rodman, and Nina Hyams. *An Introduction to Language*. Wadsworth Cengage Learning, 2011.

Jesús Giménez and Lluís Màrquez. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, Lisbon, Portugal, 2004.

Andrew S. Gordon and Reid Swanson. Generalizing semantic role annotations across syntactically similar verbs. In *In Proceedings of Association for Computational Linguistics*, pages 192–199, 2007.

Jan Hajič. Building a syntactically annotated corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19. Prague Karolinum, Charles University Press, 1998.

Katsuhioko Hayashi, Shuhei Kondo, Kevin Duh, and Yuji Matsumoto. The naist dependency parser for sancl2012 shared task. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, 2012.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/. [Online; accessed 2015-02-03].

Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 595–603, Columbus, OH, 2008.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Massachusetts Institute of Technology, 1999.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 0521865719, 9780521865715.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

Ra Kübler Ryan Mcdonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The conll 2007 shared task on dependency parsing. In *In Proceedings of CoNLL. Slav Petrov, Dipanjan*, 2007.

Ryan McDonald and Joakim Nivre. Analyzing and integrating dependency parsers. *Comput. Linguist.*, 37(1):197–230, March 2011.

Ryan Mcdonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *In Proc. of EACL*, pages 81–88, 2006.

Scott Miller, Jethran Guinness, and Alex Zamanian. Name tagging with word clusters and discriminative training. In *Proccedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 337–342, Boston, MA, 2004.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the 1st International Joint Conference on Natural Language Processing*, pages 684–693, Hainan Island, China, 2004.

Joakim Nivre. Dependency grammar and dependency parsing. Technical report, Växjö University, 2005.

Joakim Nivre and Johan Hall. A dependency-driven parser for german dependency and constituency representations. In *Proceedings of the Association for Computational Linguistics: Human Language Technology Conference Workshop on Parsing German*, pages 47–54, 2008.

Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006)*, pages 2216–2219, 2006.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, Sandra Kübler, Marinov Svetoslav, Erwin Marsi, and Atanas Chanev. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

Lilja Øvrelid and Arne Skjærholt. Lexical categories for improved parsing of web data. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 903–912, Bombay, India, 2012.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Slav Petrov and Ryan McDonald. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, Montreal, Canada, 2012.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the*

*21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, 2006.

Tony Rose, Mark Stevenson, and Miles Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *In Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31, 2002.

Kenji Sagae and Andrew S. Gordon. Clustering words by syntactic similarity improves dependency parsing of predicate-argument structures. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 192–201, Paris, France, 2009.

D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1177–1178, New York, NY, USA, 2010. ACM.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden, 2010.

Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Strictly lexical dependency parsing. In *In Proceedings of the Ninth International Workshop on Parsing Technology*, pages 152–159, 2005.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. OntoNotes release 4.0 LDC2011T03, 2011.

Arnold M. Zwicky. Heads. *Journal of Linguistics*, 21, 3 1985.